12-1-2004

Numerical Integration of Linear and Nonlinear Wave Equations

Laura Lynch
Florida Atlantic University, llynch@ccga.edu

Numerical Integration of Linear and Nonlinear Wave Equations

by

Laura Lynch

A Thesis presented to the Faculty of

The Honors College of Florida Atlantic University

In Partial Fulfillment of Requirements for the Degree of

Bachelor of Arts in Liberal Arts and Sciences

with a Concentration in Physics

Under the Supervision of Professor Mark Rupright

Harriet L. Wilkes Honors College

of Florida Atlantic University

Jupiter, Florida

December 2004

Numerical Integration of Linear and Nonlinear Wave Equations

by

Laura Lynch

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Mark Rupright, and has been approved by the members of her/his supervisory committee. It was submitted to the faculty of The Harriet L. Wilkes Honors College and was accepted in partial fulfillment of the requirements for the degree of Bachelor of Arts in Liberal Arts and Sciences.

SUPERVISORY COMMITTEE:

_____

Dr. Mark Rupright

_____

Dr. Ryan Karr

_____

Dean, Harriet L. Wilkes Honors College

_____

Date

Acknowledgments

I would like to first thank Dr. Mark Rupright, my advisor, for all of his help during a busy semester. Being his first student to complete a thesis, I think we learned alot from each other. I would also like to thank my reader Dr. Ryan Karr for reading my second undergraduate thesis in a field not of his own.

Abstract

Author:	Laura Lynch
Title:	Numerical Integration of Linear and Nonlinear Wave Equations
Institution:	Harriet L. Wilkes Honors College, Florida Atlantic University
Thesis Advisor:	Mark Rupright
Concentration:	Physics
Year:	2004

We begin our study with an analysis of various numerical methods and boundary conditions on the well-known and well-studied advection and wave equations, in particular we look at the FTCS, Lax, Lax-Wendroff, Leapfrog, and Iterated Crank Nicholson methods with periodic, outgoing, and Dirichlet boundary conditions. We will then extend our study to the nonlinear equation $g_{tt} = g_{xx} - g_t^2/g$, introduced by Khoklov and Novikov. The nonlinearities are similar to those seen in General Relativity, and thus our analysis establishes the effects of numerical integration and boundary condition choices on the long-term stability of gravitational wave simulations.

Contents

List of Tables

List of Figures

1 Introduction

Solving partial differential equations is an art that can be difficult, especially when done numerically. There are many factors that must be considered. Our goal is to pick a numerical method that not only makes sense for the equation we are solving, but one that is stable over a given amount of time, one that has a sufficiently high order accuracy, and one that minimizes the time required to compute the solution while keeping the accuracy. Also, boundary conditions must be considered as they can effect a solution drastically and may cause discontinuities.

Applications in numerical partial differential equations fall under gravitational wave research and numerical relativity, to name a few. Current research in the study of gravitational waves includes projects like LIGO (Laser Interferometer Gravitational-Wave Observatory) which are looking to find the existence of gravity waves (see [BW], [W], or [KT] for more information). In finding gravity waves, we require a way to model them in order to understand exactly what is happening. Numerical methods are required to determine whether gravity waves are produced by such events as black holes colliding.

In Chapter 2, we present the necessary background information to conduct our study. We will look briefly at some analytical solutions for a partial differential equation, including a section on the characteristic equation and the D'Alembert solution. Chapter 2 also presents tools for numerical methods, including numerical differentiation, order of accuracy, and norms.

Chapter 3 begins our study of numerical methods on partial differential equations, where we look at the advection equation. The advection equation, namely $a_t = \pm c a_x$, is a well-known partial differential equation with known solutions. As such, we can compare the various numerical methods we investigate with the known solution for strengths and weaknesses.

Chapter 4 takes our study on a deeper level by analyzing various numerical methods for the one-dimensional linear wave equation. The wave equation, $u_{tt} = c^2 u_{xx}$, is a second-order partial differential equation (unlike the first-order advection equation) and is more synonymous with the nonlinear equations we will be looking at in Chapter 6. Once we have gained an understanding of the various numerical methods and how they affect a given solution, we move on to the effects of boundary conditions in Chapter 5. Chapter 5 considers outgoing and Dirichlet boundary conditions for the wave equation as opposed to the periodic boundary conditions found in Chapters 3 and 4.

Equipped with the knowledge of using the various numerical methods and imposing the various boundary conditions, we consider a nonlinear system without a general solution in Chapter 6. The nonlinear system, namely $g_{tt} = g_{xx} - g_t^2/g$, was studied in a paper by Hansen, Khokhlov, and Novikov, where they considered the stability of four numerical methods, namely the third and fourth order Runge-Kutta methods, the Courant-Friedrichs-Levy Method, and the Iterated Crank-Nicholson method. We will consider the equation with other numerical methods and compute the numerical value rather than performing a stability analysis. In Chapter 7, we

will consider the possibilities for future research for our nonlinear system.

2 Background

A partial differential equation, or PDE for short, in its simplest terms is a differential equation involving partial derivatives. The study of partial differential equations can be difficult as unknown functions are dependent on more than one variable, unlike in ordinary differential equations. Linear PDEs (those where the dependent variable and its derivatives are not multiplied together) are classified into three basic types: parabolic, hyperbolic, and elliptic. For the general second-order linear equation in two-variables with constant coefficients,

$$Au_{xx} + Bu_{xt} + Cu_{tt} + Du_x + Eu_t + Fu + G = 0 \ ,$$

where

$$u_t = \frac{\partial u}{\partial t}$$

and

$$u_{xt} = \frac{\partial}{\partial x}\frac{\partial u}{\partial t} \ .$$

The type of classification depends on the constants A, B, and C. That is, if $B^2 - 4AC = 0$ then the equation is parabolic, if $B^2 - 4AC > 0$ then the equation is hyperbolic, and if $B^2 - 4AC < 0$ the equation is elliptic. In this paper, we will deal only with hyperbolic PDEs and so the following analysis will focus on them.

2.1 Analytical Solutions

In hyperbolic partial differential equations, one of the independent variables is usually time or analogous to time. There are various ways to analytically solve a hyperbolic partial differential equation, including the most basic of ways: Separation of Variables (a technique students learn in Calculus II). Regardless of which technique is used, initial and/or boundary conditions of the unknown function are always necessary to get a specific solution to the PDE.

2.1.1 Initial and Boundary Conditions

Initial conditions specify the unknown function, and possibly its derivatives, at an initial time. Various equations require a different number of initial conditions, depending on the order of the time derivative(s). The advection equation, $a_t = -ca_x$, for example requires only one initial condition (on $a(x,t)$) since it has only first-order derivatives in time. In contrast, the wave equation, $u_{tt} = c^2 u_{xx}$, requires two initial conditions– one on $u(x,t)$ and one on $u_t(x,t)$ since it has a second-order derivative in time.

Along with initial conditions, boundary conditions are needed for bounded domains if we want to solve for u at all future times. Consider Figure 1, which shows the domain of dependence of initial and boundary conditions for a typical hyperbolic partial differential equation.

The two vertical dashed lines are the boundaries of the grid. The initial condition

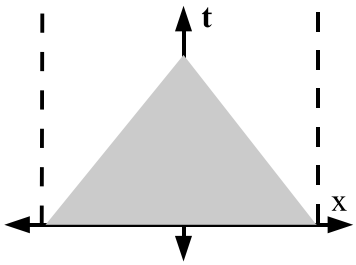

Figure 1: How initial conditions affect solutions

is specified between the boundaries on the x-axis. The shaded region comprises all points where u is governed solely by initial conditions. At points in the non-shaded region, the value of u depends on boundary conditions as well as initial conditions. There are many types of boundary conditions that can be applied to the function and the choice of boundary conditions depends on physical considerations as well as analytical and numerical properties.

Some examples that we will use in this paper include Dirichlet boundary conditions which use the actual value of the function at that position (this only works for equations with known solutions at the boundary), and outgoing boundary conditions that involve only points within the grid and do not involve waves that may come into the grid from outside. We are also interested in periodic boundary conditions which equate the values of the function at the boundary. Then our wave exists on the surface of a cylinder where the initial wave is specified on the circumference of the bottom of the cylinder, as in Figure 2. It is important to notice that the domain of the wave is only on the surface of the cylinder and not on the inside and that by equating the spatial boundaries there actually is no longer a spatial boundary.

Using periodic boundary conditions allow us to study the evolution of the wave as

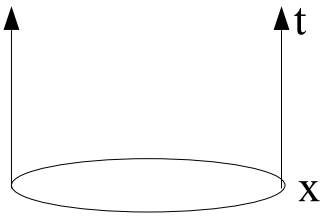

Figure 2: Periodic Boundary Conditions create a cylinder

a separate analysis from boundary conditions.

2.1.2 Characteristic Equations

In solving partial differential equations, especially those where separation of variables does not work, it is often easier to transform the equation into the *canonical form* using what are called *characteristic equations* so that, for example, an equation of three second-order derivatives can be simplified to an equation of one second-order derivative.

Consider the general equation mentioned earlier:

$$Au_{xx} + Bu_{xt} + Cu_{tt} + Du_x + Eu_t + Fu + G = 0.$$

To simplify this equation into canonical form, we want to do a change of variables from x, t to $\xi = \xi(x,t), \tau = \tau(x,t)$. First, we must compute all of the partial

derivatives

$$\begin{aligned}
u_x &= u_\xi \xi_x + u_\tau \tau_x \\
u_t &= u_\xi \xi_t + u_\tau \tau_t \\
u_{xx} &= u_{\xi\xi} \xi_x^2 + u_{\xi\tau} \xi_x \tau_x + u_{\tau\tau} \tau_x^2 + u_\xi \xi_{xx} + u_\tau \tau_{xx} \\
u_{xt} &= u_{\xi\xi} \xi_x \xi_t + u_{\xi\tau} \left(\xi_x \tau_t + \xi_t \tau_x \right) + u_{\tau\tau} \tau_x \tau_t + u_\xi \xi_{xt} + u_\tau \tau_{xt} \\
u_{tt} &= u_{\xi\xi} \xi_t^2 + u_{\xi\tau} \xi_t \tau_t + u_{\tau\tau} \tau_t^2 + u_\xi \xi_{tt} + u_\tau \tau_{tt}
\end{aligned}$$

and then substitute them back into our original PDE. Simplifying, we end up with

$$a u_{\xi\xi} + b u_{\xi\tau} + c u_{\tau\tau} + d u_\xi + e u_\tau + f u + g = 0 \, ,$$

where

$$\begin{aligned}
a &= A \xi_x^2 + B \xi_x \xi_t + C \xi_t^2 \\
b &= 2A \xi_x \tau_x + B \left(\xi_x \tau_t + \xi_t \tau_x \right) + 2C \xi_t \tau_t \\
c &= A \tau_x^2 + B \tau_x \tau_t + C \tau_t^2 \\
d &= A \xi_{xx} + B \xi_{xt} + C \xi_{tt} + D \xi_x + E \xi_t \\
e &= A \tau_{xx} + B \tau_{xt} + C \tau_{tt} + D \tau_x + E \tau_t \\
f &= F \\
g &= G \, .
\end{aligned}$$

This may not seem simpler than our original equation, but we are not finished. Since we have not specified what the functions ξ and τ are yet, we can choose them so that a and c are zero. Then, the above equation simplifies to the canonical form

$$b u_{\xi\tau} + d u_\xi + e u_\tau + f u + g = 0 \, ,$$

where there is only one second-order derivative (instead of three) and

$$\begin{aligned} 0 &= A\xi_x^2 + B\xi_x\xi_t + C\xi_t^2 \\ 0 &= A\tau_x^2 + B\tau_x\tau_t + C\tau_t^2 \ . \end{aligned}$$

We can solve for the ratios ξ_x/ξ_t and τ_x/τ_t using the quadratic formula as follows

$$\begin{aligned} \xi_x/\xi_t &= \frac{-B + \sqrt{B^2 - 4AC}}{2A} \\ \tau_x/\tau_t &= \frac{-B - \sqrt{B^2 - 4AC}}{2A} \end{aligned}$$

These two equations are known as the *characteristic equations* as they help us to find the *characteristics* of our PDE, that is, the curves (or surfaces) for which our *characteristic coordinates* ξ and τ are constant. How does this work? To find where $\xi(x,t)$ and $\tau(x,t)$ are constant, note that

$$d\xi = \xi_x dx + \xi_t dt = 0$$

and

$$d\tau = \tau_x dx + \tau_t dt = 0$$

which implies

$$\frac{dt}{dx} = -\frac{\xi_x}{\xi_t}$$

and

$$\frac{dt}{dx} = -\frac{\tau_x}{\tau_t}.$$

Thus by the characteristic equations, we have

$$\begin{aligned} \frac{dt}{dx} &= \frac{B - \sqrt{B^2 - 4AC}}{2A} \ , \text{ and} \\ \frac{dt}{dx} &= \frac{B + \sqrt{B^2 - 4AC}}{2A} \ . \end{aligned} \tag{1}$$

From here, we can integrate to find ξ and τ by equating them to the constants of integration. Now, all that is needed is to plug the characteristic coordinates into the PDE to arrive at the Canonical form.

We will consider the wave equation as an example of how characteristics are used for solving PDEs in the next section.

2.1.3 D'Alembert Solution

With a PDE in canonical form, we can use the D'Alembert solution to solve the PDE (with initial conditions). Consider the wave equation $u_{tt} = c^2 u_{xx}$. From equation (1), since $A = -c^2, B = 0, C = 1$ we see that

$$\begin{aligned} \frac{dt}{dx} &= \frac{B - \sqrt{B^2 - 4AC}}{2A} = -\frac{1}{c} \\ \frac{dt}{dx} &= \frac{B + \sqrt{B^2 - 4AC}}{2A} = \frac{1}{c}. \end{aligned} \tag{2}$$

Integrating, we get $x = -ct + \xi$ and $x = ct + \tau$ and thus $\xi = x + ct$ and $\tau = x - ct$. Substituting these into the coefficients of our transformed equation

$$bu_{\xi\tau} + du_{\xi} + eu_{\tau} + fu + g = 0$$

we have

$$\begin{aligned} b &= 2A\xi_x\tau_x + B\left(\xi_x\tau_t + \xi_t\tau_x\right) + 2C\xi_t\tau_t = 2(-c^2) + 2(c)(-c) = -4c^2 \\ d &= A\xi_{xx} + B\xi_{xt} + C\xi_{tt} + D\xi_x + E\xi_t = 0 \\ e &= A\tau_{xx} + B\tau_{xt} + C\tau_{tt} + D\tau_x + E\tau_t = 0 \\ f &= F = 0 \\ g &= G = 0 \end{aligned}$$

thus $-4c^2 u_{\xi\tau} = 0$ and we arrive at the canonical form

$$u_{\xi\tau} = 0.$$

Now, we can simply integrate the equation twice to get $u(\xi, \tau) = f(\tau) + g(\xi)$. Substituting x and t back, we get

$$u(x, t) = f(x - ct) + g(x + ct)$$

which is the general solution for the wave equation. Here, the function $f(x - ct)$ defines right-moving waves and the function $g(x + ct)$ defines left-moving waves. The initial data is thus being dragged along the constant lines of $x + ct$ and $x - ct$.

2.2 Order of Accuracy

According to [NM], the order of accuracy is defined as follows.

Definition 2.1. *The function $f(h)$ is said to be $O(g(h))$ if there exists a constant C such that $|f(h)| \leq C|g(h)|$ for sufficiently large h. We write $f(h) = O(g(h))$.*

An example, and where $O(h)$ is probably seen most frequently, is the Taylor series expansion of $f(x)$, namely

$$f(x + h) = \sum_{k=0}^{n} \frac{f^{(k)}(x)}{k!}(h)^k + \frac{f^{(n+1)}(\xi)}{(n+1)!}(h)^{n+1},$$

for some $\xi \in (x, x + h)$. Here, the order of accuracy is simply the smallest power of h that was truncated, that is,

$$f(x + h) - \sum_{k=0}^{n} \frac{f^{(k)}(x)}{k!}(h)^k = O(h^{n+1}).$$

The order of accuracy is the "truncation error" or the difference between the true function and the Taylor series "truncated" at n. This tells us that the error in the Taylor series approximation for small h will primarily be proportional to h^{n+1}.

2.3 Numerical Differentiation

The derivative a function is defined as

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

or

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}.$$

Using the Taylor series, we can actually use these equations to find numerical approximations of the derivative. Consider the former equation for the derivative,

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

The Taylor series yields

$$f(x+h) = f(x) + f'(x)h + O(h^2).$$

Rewriting, we see

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h).$$

Notice that since we divided by h, our order of accuracy decreased by one power of h. Thus we can approximate the derivative by

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

with accuracy of order h. Notice that if we cut h in half, we would expect the error in our approximation to be half as much as before. This is called the one-sided difference formula as it only considers values of the function on one side of x.

Similarly, if we consider our second function for a derivative, namely

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}.$$

The Taylor series now yields

$$f(x+h) = f(x) + f'(x)h + \frac{f^{(2)}(x)h^2}{2!} + O(h^3)$$

and

$$f(x-h) = f(x) - f'(x)h + \frac{f^{(2)}(x)h^2}{2!} + O(h^3).$$

Thus

$$f(x+h) - f(x-h) = 2f'(x)h + O(h^3).$$

Dividing by $2h$ and rewriting, we see

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2).$$

Thus we can approximate the derivative by

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

with accuracy of order h^2, that is, if we decrease h by one half, the error in our approximation will be one fourth of the original error. This is called the centered difference formula as values of the function are centered around x. It is important

to notice that the one sided difference formula is less accurate than the centered difference formula. This fact will become important later in our study.

In a similar manner, we can approximate the second derivative of a function. It seems obvious to use the centered difference (with its higher order of accuracy) twice. We see

$$f'\left(x+\frac{h}{2}\right) \approx \frac{f(x+h)-f(x)}{h}$$

and

$$f'\left(x-\frac{h}{2}\right) \approx \frac{f(x)-f(x-h)}{h}\ .$$

Since

$$f''(x) \approx \frac{f'(x+\frac{h}{2})-f'(x-\frac{h}{2})}{h}\ ,$$

we see

$$f''(x) \approx \frac{f(x+h)-2f(x)+f(x-h)}{h^2}\ .$$

One can show that this has accuracy of order h^2.

2.4 Norms

The norm of a vector $\mathbf{x}$, written $||\mathbf{x}||$, is a function that satisfies the following three conditions.

1. $||\mathbf{x}|| > 0$ for all nonzero vectors $\mathbf{x}$.

2. $||a\mathbf{x}|| = |a|||\mathbf{x}||$ for all vectors $\mathbf{x}$ and scalars a.

3. $||\mathbf{x}+\mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||$ for all vectors $\mathbf{x}$ and $\mathbf{y}$.

In general, norms are used to find the "size" of a vector. We will use norms to calculate the "size" of the error between our numerical solutions (defined on a one-dimensional grid) and the analytical solutions. The simplest, and most common, norms are the L_p norms and they are defined as follows:

$$||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}.$$

Examples of the L_p norms are

1. L_1 Norm: $||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i|$,

2. L_2 Norm: $||\mathbf{x}||_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$,

3. L_∞ Norm: $||\mathbf{x}||_\infty = \max\{|x_1|, |x_2|, ..., |x_n|\}$.

Throughout this paper, we will use the L_2 norm as it is the most commonly used.

3 Advection Equation

The advection equation,

$$\frac{\partial a}{\partial t} = -c\frac{\partial a}{\partial x}$$

has the general solution $a(x,t) = f(x - ct)$. Graphically, as t increases, the initial function $a(x,0)$ will move to the right with speed c.[1] We will solve the equation in the particular case where the initial condition is the cosine-modulated Gaussian pulse,

$$a(x,0) = \cos(kx)\exp\left(\frac{x^2}{2\sigma^2}\right)$$

with periodic boundary conditions:

$$a\left(-\frac{L}{2},t\right) = a\left(\frac{L}{2},t\right)$$

and

$$\frac{da}{dx}\Big|_{x=-L/2} = \frac{da}{dx}\Big|_{x=L/2}.$$

We use the cosine-modulated Gaussian pulse because it is smooth (and therefore differentiable) and numerically compact (the wave has finite width and outside that width, values are approximately zero). We multiply by the cosine, in particular, because it has a larger second derivative than does a Gaussian. We use periodic boundary conditions as it lets our wave remain in the domain forever (presenting a greater challenge in terms of finite difference error). Periodic boundary conditions allow us to see what a particular numerical method does to a solution in the long run.

[1] The advection equation for a left-moving wave is given by $\frac{\partial a}{\partial t} = c\frac{\partial a}{\partial x}$.

In solving, we will look at the derivatives in their discretized form. Using this strategy, we can easily compute the solution one time step at a time, or let a program like MATLAB do it for us. For each method studied below, we will first describe how it works, show the method in action using pictures created in MATLAB (see the Appendix for MATLAB code), then explain the benefits and weaknesses.

For simplicity, all numerical solutions will be calculated with $c = 1$ and $L = 1$, where L is the width of the grid. Also, we will divide the width of the grid into 50 equal parts, or "zones" (thus the grid spacing is $h = \frac{1}{50} = .02$, where h is the spatial step). We want the total time for our algorithm to be the time it takes the wave to move one complete grid length, namely $c\tau = L$. This is simply the time it takes for the wave to reach its initial position: one crossing time. (Recall that the boundary conditions are periodic, so as the wave moves off the grid on one side, it will reappear on the other.) Thus the only variable we adjust is τ, since c, L, and h are held constant.

3.1 FTCS Method

3.1.1 How it works

For the FTCS (Forward Time, Centered Space) method, we approximate the time derivative in the advection equation with the forward discretized form

$$\frac{\partial a}{\partial t} \approx \frac{a_i^{n+1} - a_i^n}{\tau} \ ,$$

where $a_i^n = a(x_i, t_n)$ and $\tau = t_{n+1} - t_n$ is our time step. We then approximate the spatial derivative with the centered discretized form

$$\frac{\partial a}{\partial x} \approx \frac{a_{i+1}^n - a_{i-1}^n}{2h} ,$$

where $h = x_i - x_{i-1}$ is again our spatial grid spacing. Substituting the discretized forms of the derivatives into the advection equation yields the approximation

$$\frac{a_i^{n+1} - a_i^n}{\tau} = -c\frac{a_{i+1}^n - a_{i-1}^n}{2h}$$

which implies

$$a_i^{n+1} = a_i^n - \frac{c\tau}{2h}(a_{i+1}^n - a_{i-1}^n) .$$

Using the equation above, we explicitly solve for a^{n+1} using the values of a from the previous time step n. Thus we can find an approximation for the wave at any point knowing only the initial position of the wave. Note, however, that we cannot use the equation to compute a_1^{n+1} (thus the need for boundary conditions). It is often easier to draw a "computational molecule" to show exactly how the equation works and how the points are related. We present this depiction in Figure 3.

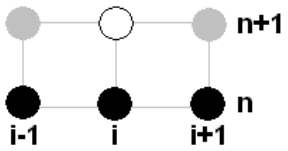

Figure 3: Computational Molecule for FTCS Method

Each point represents a value of the function a. The black points are the known data that we use to evaluate the function at the white point and the grey points are unused.

3.1.2 Stability

The von Neumann stability analysis is a way to determine when a particular numerical method is stable. It looks at solutions of the form $a_j^n = \xi^n e^{ijkh}$, where $i = \sqrt{-1}$, j is our spatial index, k is the time index, and h is the spatial step. To do the analysis, we simply substitute the above solution into the discretized form of the numerical method and determine where $|\xi|^2 \leq 1$. This tells us where the amplitude of the wave is less than or equal to one. If the amplitude is greater than one, then the amplitude is increasing and will therefore eventually become unstable. Thus the method is stable at the values where $|\xi|^2 \leq 1$.

Consider the FTCS method, where the discretized equation is

$$a_j^{n+1} = a_j^n - \frac{c\tau}{2h}(a_{j+1}^n - a_{j-1}^n)$$

(notice we are using j for the spatial index instead of i to avoid confusion). Substituting in $a_j^n = \xi^n e^{ijkh}$, we get

$$\xi^{n+1} e^{ijkh} = \xi^n e^{ijkh} - \frac{c\tau}{2h}(\xi^n e^{i(j+1)kh} - \xi^n e^{i(j-1)kh}).$$

Diving both sides of the equation by ξ^n and e^{ijkh} yields

$$\xi = 1 - \frac{c\tau}{2h}(e^{ikh} - e^{-ikh}).$$

By a well-known property of trigonometric functions, we have

$$\xi = 1 - \frac{c\tau i}{h}(\sin(kh))$$

which implies

$$|\xi|^2 = 1 + \left(\frac{c\tau i}{h}\right)^2 \sin^2(kh).$$

Therefore, $|\xi|^2$ is always larger than 1 and the FTCS method is unconditionally unstable.

3.1.3 Solutions

Even though the von Neuman stability analysis shows that the FTCS method is unstable, lets look at the graph to see what happens. We expect that it will fairly quickly tend to infinity, that is, it will "blow up." We will look at the particular case of $\tau = h$. This simplifies our equation to $a_i^{n+1} = a_i^n - \frac{1}{2}(a_{i+1}^n - a_{i-1}^n)$. Looking at Figure 4, we see that our method blows up drastically after only 10 steps ($\frac{1}{5}$ of one crossing time).

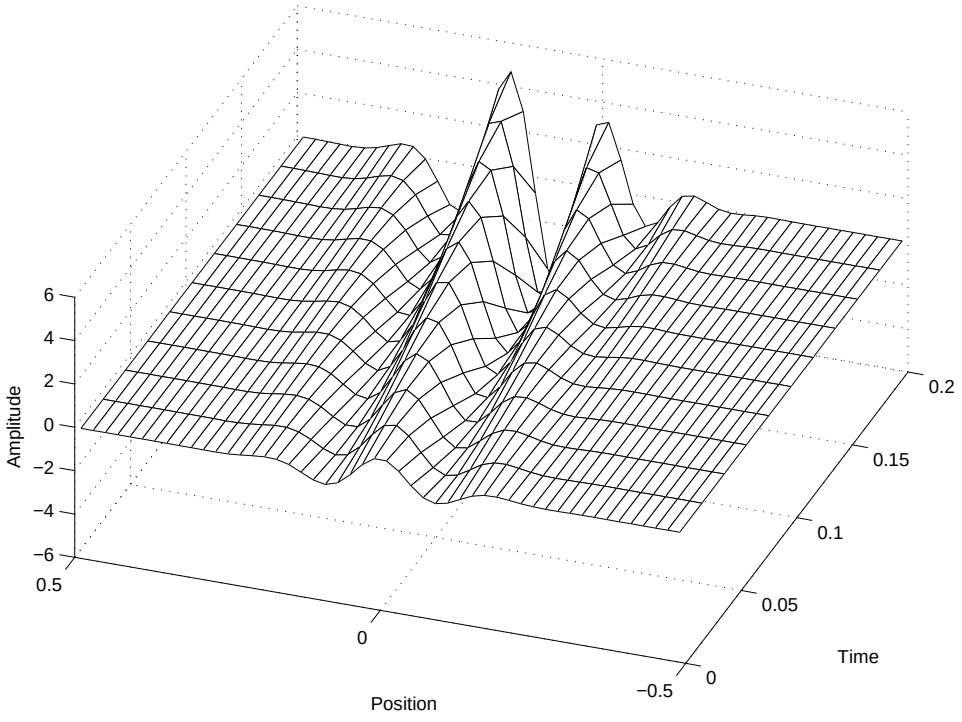

Figure 4: FTCS Method for Advection Equation with $\tau = h$.

The amplitude, which should be constant, appears to grow without bound. To see how quickly the error in the numerical solution grows, we want to look at an error plot. Figure 5 is a log-log plot of the error between the analytical solution and the numerical solution calculated through FTCS. If the growth of the error were linear on this plot, then the error would grow as a power law. Figure 5 shows us that FTCS blows up *faster* than a power-law function.

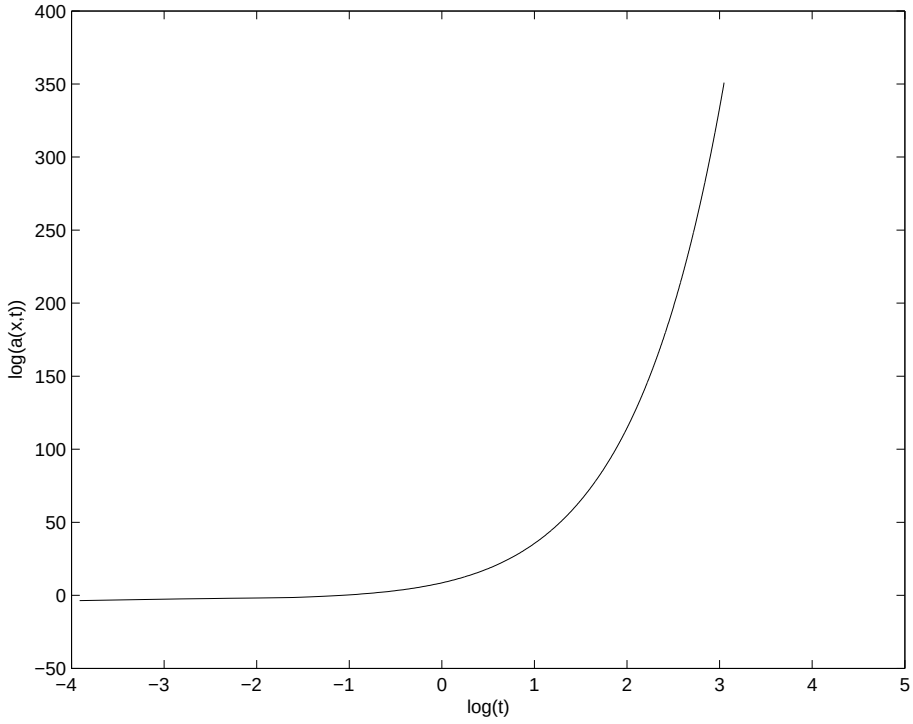

Figure 5: Log-Log plot of FTCS Method for Advection Equation with $\tau = h$.

3.2 Upwind Method

The FTCS method used a centered space difference which, in general, is more accurate than a one sided difference due to the $O(h^n)$ error. However, we are ap-

proximating a one-directional wave. Thus using data from the left and right of the evaluation point is not useful because the wave only comes from one of those directions. One of those points should not affect our solution. So we will now take a step backward and look at a one-sided difference for both the time and space derivatives.

3.2.1 How it works

The Upwind method solves the advection equation by approximating the time derivative with the forward (right) discretized form,

$$\frac{\partial a}{\partial t} \approx \frac{a_i^{n+1} - a_i^n}{\tau} \ ,$$

and the spatial derivative with the backward (left) discretized form

$$\frac{\partial a}{\partial t} \approx \frac{a_i^n - a_{i-1}^n}{h} \ .$$

We chose this because our wave moves to the right. To find the value at x_i, we need the data at x_{i-1}. Note that if we were looking at the left-moving advection equation, that is, if the advection equation had the opposite sign, we would have to use data at x_{i+1}. Then we approximate the PDE by the "difference equation"

$$\frac{a_i^{n+1} - a_i^n}{\tau} = -c\frac{a_i^n - a_{i-1}^n}{h}$$

which implies

$$a_i^{n+1} = a_i^n - \frac{c\tau}{h}(a_i^n - a_{i-1}^n).$$

As with the FTCS method, we cannot compute the boundary points using this equation so we must again use boundary conditions.[2] For a graphical version, see Figure 6.

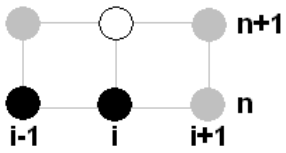

Figure 6: Computational Molecule for Upwind Method

Again, each point represents a value of the function a. The black points are the ones used in the equation and the grey points are just other points found on the grid.

3.2.2 Stability

In the previous section, we saw that the FTCS method was unconditionally unstable. Before computing solutions, we again want to consider the stability of our system. Following the procedures of the von Neumann stability analysis, we want to substitute $a_j^n = \xi^n e^{ijkh}$ into our discretized equation

$$a_j^{n+1} = a_j^n - \frac{c\tau}{h}(a_j^n - a_{j-1}^n)$$

(again notice we changed the index from i to j). This gives us

$$\xi^{n+1} e^{ijkh} = \xi^n e^{ijkh} - \frac{c\tau}{h}(\xi^n e^{ijkh} - \xi^n e^{i(j-1)kh}).$$

[2]Note that a uni-directional wave only needs one boundary condition, as does a uni-directional scheme.

Dividing both sides by ξ^n and e^{ijkh} simplifies our equation to

$$\xi = 1 - \frac{c\tau}{h}(1 - e^{-ikh})$$

or

$$\xi = \left(1 - \frac{c\tau}{h}\right) + \left(\frac{c\tau}{h}\right) e^{-ikh}.$$

To show stability, we want to find conditions so that $0 \leq |\xi|^2 \leq 1$. We know that $|\xi|^2$ is just

$$\begin{aligned}
|\xi|^2 &= \left[\left(1 - \tfrac{c\tau}{h}\right) + \left(\tfrac{c\tau}{h}\right) e^{-ikh}\right]\left[\left(1 - \tfrac{c\tau}{h}\right) + \left(\tfrac{c\tau}{h}\right) e^{ikh}\right] \\
&= \left(1 - \tfrac{c\tau}{h}\right)^2 + \left(1 - \tfrac{c\tau}{h}\right)\left(\tfrac{c\tau}{h}\right)(e^{-ikh} + e^{ikh}) + \left(\tfrac{c\tau}{h}\right)^2 \\
&= 1 - 2\left(\tfrac{c\tau}{h}\right) + 2\left(\tfrac{c\tau}{h}\right)^2 + 2\left(1 - \tfrac{c\tau}{h}\right)\left(\tfrac{c\tau}{h}\right)\cos(kh) \\
&= 1 - 2\left(1 - \tfrac{c\tau}{h}\right)\left(\tfrac{c\tau}{h}\right)[1 - \cos(kh)]
\end{aligned}$$

To see where $0 \leq |\xi|^2 \leq 1$, we can simply show

$$0 \leq 2\left(1 - \frac{c\tau}{h}\right)\left(\frac{c\tau}{h}\right)[1 - \cos(kh)] \leq 1.$$

Since $1 - \cos(kh)$ is always greater than or equal to zero, we require $1 - \frac{c\tau}{h}$ must also be greater than zero, that is,

$$1 \geq \frac{c\tau}{h}.$$

So as long as this condition is true, the Upwind method is stable. This is actually known as the Courant condition, or the Courant-Friedrichs-Lewy (CFL) condition and the fraction $\frac{c\tau}{h}$ is called a Courant factor. For explicit schemes (that is, those that do not require data from the $(n+1)$st time step), this condition requires that

the time step, τ, be smaller than the time it takes for the wave to move to adjacent points, $\frac{h}{c}$.

In all future calculations, we will look at different Courant factors, and in particular, different Courant factors less than 1. For simplicity, we will call the Courant factor α. Thus $\alpha = \frac{c\tau}{h}$.

3.2.3 Solutions

To see our model in action, we first want to check the trivial case, $\alpha = 1$ or $\tau = h = .02$. This is trivial only because $h = c\tau$, which simplifies the equation to $a_i^{n+1} = a_i^n - \frac{c\tau}{h}(a_i^n - a_{i-1}^n) = a_{i-1}^n$. Thus the wave is simply being dragged along without any dampening, that is, without any dissipation factors. Analytically, the trivial solution will give a final wave equivalent to the initial wave since their is no dissipation (dampening) or dispersion (shape changing) factors. Using the MATLAB code outlined in the Appendix, we can compute the wave using our equation for the Upwind method and our given time step. Consider Figure 7, which shows the initial and final position of the wave as determined through MATLAB.

Since the two instances of the wave are identical, our method works for the trivial case, that is, the numerical solution we calculated is equivalent to the analytical solution. Now we want to consider other values of α. Looking at Figure 8, we see that our method severely dampens the wave between the initial and final times as our Courant factor gets smaller. (Note: For $\alpha > 1$, the Upwind method produces garbage because it is unstable.)

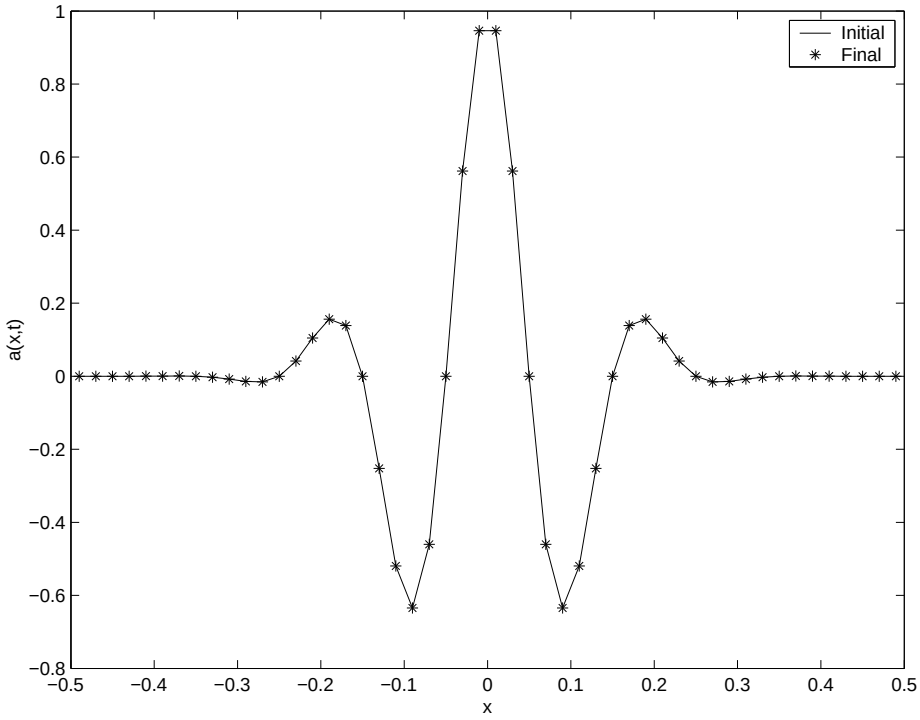

Figure 7: Upwind Method for Advection Equation with $\alpha = 1$.

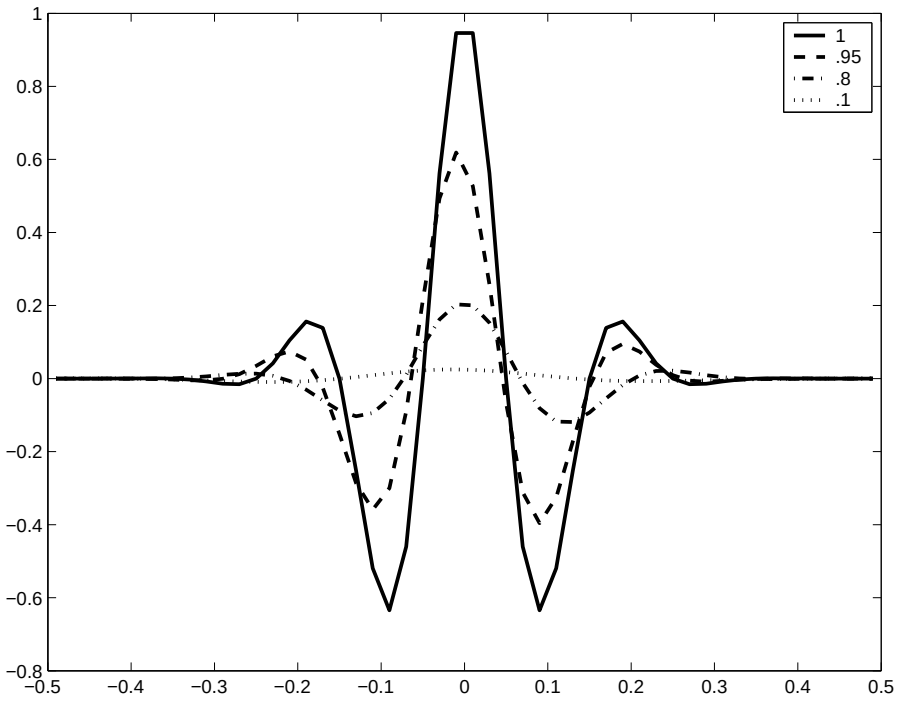

Figure 8: Upwind Method for Advection Equation with various α.

The advantage of the Upwind method is its simplicity. However, the severe dissipation and first-order error in time and space make this method less desirable. In addition, as we stated before, Upwind only works for waves moving in one direction (since it only takes into account the forward time, backward space differences) and cannot be used for second-order wave equations. Therefore, we will consider other methods.

3.3 Lax Method

3.3.1 How it works

The Lax method is fairly similar to the FTCS method, except it replaces a_i^n with the average of its neighboring points. This change actually makes the Lax method stable for certain values of α as we will see later. Thus

$$a_i^{n+1} = \frac{1}{2}(a_{i+1}^n + a_{i-1}^n) - \frac{c\tau}{2h}(a_{i+1}^n - a_{i-1}^n).$$

For a graphical version, see Figure 9.

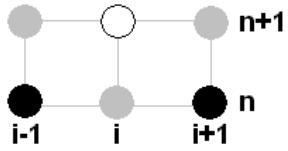

Figure 9: Computational Molecule for Lax Method

If we manipulate this equation a bit, we see that this can be rewritten as

$$\frac{a_i^{n+1} - a_i^n}{\tau} = -c\left(\frac{a_{i+1}^n - a_{i-1}^n}{2h} - \frac{h^2}{2c\tau}\left(\frac{a_{i+1}^n - 2a_i^n + a_{i-1}^n}{h^2}\right)\right).$$

This is equivalent to the FTCS approximation of

$$\frac{\partial a}{\partial t} = -c\frac{\partial a}{\partial x} + \frac{h^2}{2\tau}\frac{\partial^2 a}{\partial x^2},$$

known as the "advection diffusion" equation. According to [NR, p. 829], the last term is a dissipation factor for the advection equation. Thus by averaging the neighboring points of a_i^n, we are actually adding a dissipation term. This should cause the wave to decrease in amplitude. Another way of looking at this, however, is that the FTCS method is stable for this advection diffusion equation.

3.3.2 Stability

We again want to know the parameters (that is, the stable Courant factors) for stability in our method. As usual, we will substitute $a_j^n = \xi^n e^{ijkh}$ into our discretized equation

$$a_j^{n+1} = \frac{1}{2}(a_{j+1}^n + a_{j-1}^n) - \frac{c\tau}{2h}(a_{j+1}^n - a_{j-1}^n).$$

This gives us

$$\xi^{n+1}e^{ijkh} = \frac{1}{2}(\xi^n e^{ij+1kh} + \xi^n e^{ij-1kh}) - \frac{c\tau}{2h}(\xi^n e^{i(j+1)kh} - \xi^n e^{i(j-1)kh}).$$

Dividing both sides by ξ^n and e^{ikjh} yields

$$\xi = \frac{1}{2}(e^{ikh} + e^{-ikh}) - \frac{c\tau}{2h}(e^{ikh} - e^{-ikh}).$$

Substituting in the trigonometric functions simplifies our equation to

$$\xi = \cos(kh) - \frac{ic\tau}{h}\sin(kh).$$

Thus

$$|\xi|^2 = \cos^2(kh) + \left(\frac{c\tau}{h}\right)^2 \sin^2(kh)$$

or

$$|\xi|^2 = 1 + \left(\left(\frac{c\tau}{h}\right)^2 - 1\right)\sin^2(kh).$$

Since $\sin^2(kh) \leq 1$ always, we just want $\left(\frac{c\tau}{h}\right)^2 - 1 \leq 0$, that is, $\frac{c\tau}{h} = 1$. As with the Upwind method, our restriction is simply the Courant condition. Thus, the Lax method is stable for all $\alpha \leq 1$. From now on, we will consider only stable methods.

3.3.3 Solutions

Let us again start with the trivial case of $\alpha = 1$ (note that this is still trivial as $a_i^{n+1} = a_{i-1}^n$ for our new approximation). Looking at Figure 10, we see that our method perfectly approximated the solution.

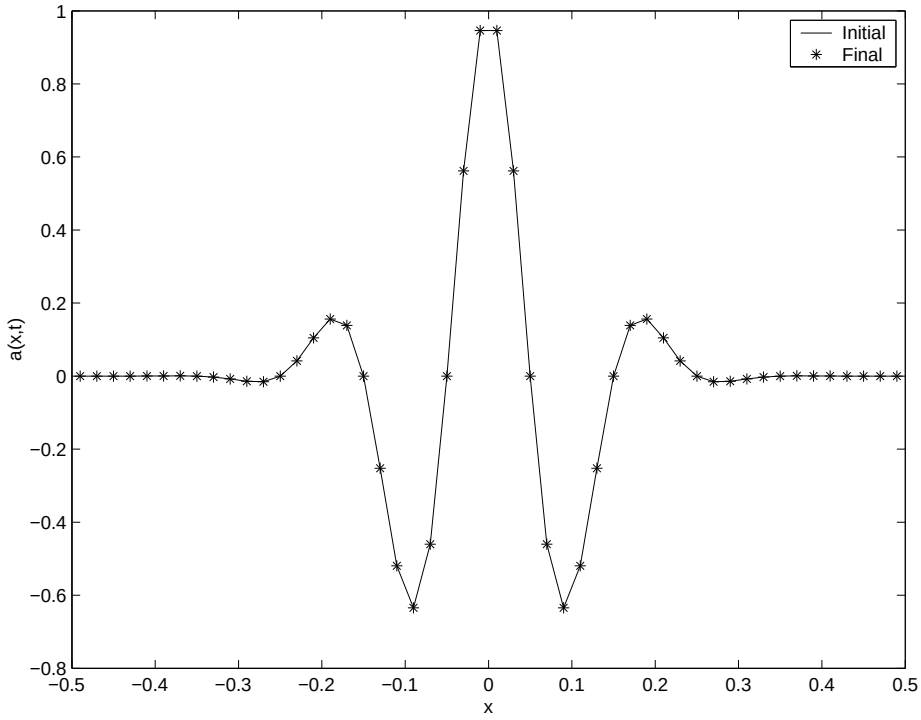

Figure 10: Lax Method for Advection Equation with $\alpha = 1$.

Looking at various Courant factors as in Figure 11, we see that this method dampens the solution even more than the Upwind method did.

It turns out that the Lax method has first-order error in time, like the Upwind method, since we used the forward time difference (instead of the centered difference). The Upwind method works better in this case, however, because the Lax method adds a larger dissipation factor which causes the wave to dampen more

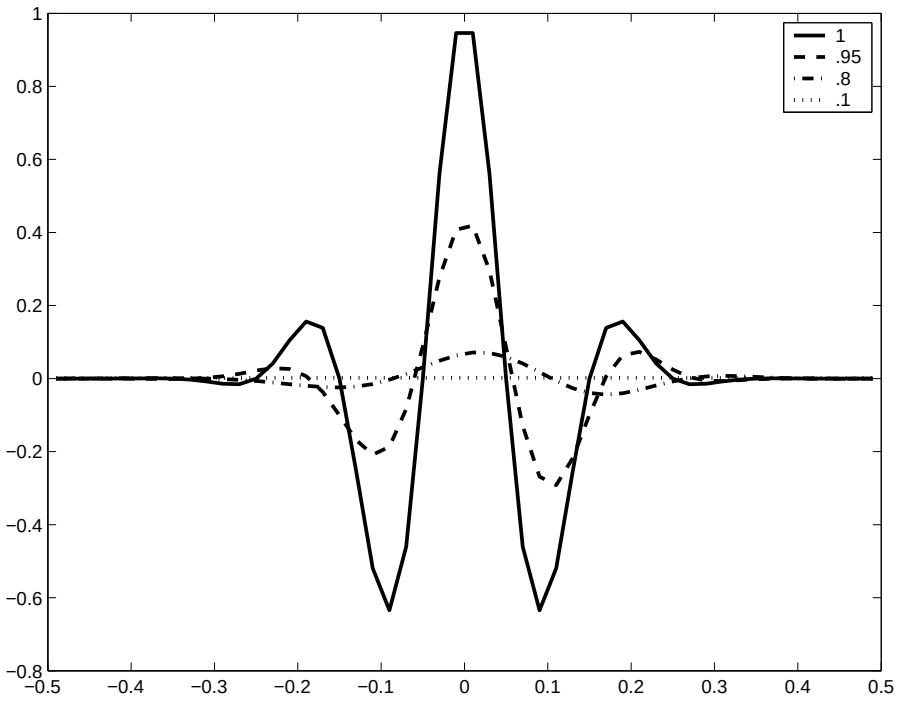

Figure 11: Lax Method for Advection Equation with various α.

than in the Upwind method. Let us try a method with second-order accuracy in time.

3.4 Lax-Wendroff Method

3.4.1 How it works

Unlike the Lax method, the Lax-Wendroff method is a second-order difference method in both time and space. For this method, we will approximate our function by the Taylor expansions (through the second-order term) where the time derivatives are replaced by the discretized centered differences. Consider the Taylor expansion

$$a(x, t+\tau) = a(x,t) + \tau\frac{\partial a}{\partial t} + \frac{\tau^2}{2}\frac{\partial^2 a}{\partial t^2} + O(\tau^3).$$

From the advection equation, we have

$$\frac{\partial a}{\partial t} = -c\frac{\partial a}{\partial x}$$

and

$$\frac{\partial^2 a}{\partial t^2} = \frac{\partial}{\partial t}\left(-c\frac{\partial a}{\partial x}\right) = c^2\frac{\partial^2 a}{\partial x^2}.$$

So, to second-order,

$$a(x, t+\tau) = a(x,t) - c\tau\frac{\partial a}{\partial x} + \frac{c^2\tau^2}{2}\frac{\partial^2 a}{\partial x^2}$$

and thus

$$a_i^{n+1} = a_i^n - \frac{c\tau}{2h}(a_{i+1}^n - a_{i-1}^n) + \frac{c^2\tau^2}{2h^2}(a_{i+1}^n - 2a_i^n + a_{i-1}^n)$$

is the approximation of the advection equation. This is the second-order (in space) discretization of the second-order (in time) series approximation. For a graphical version, see Figure 12.

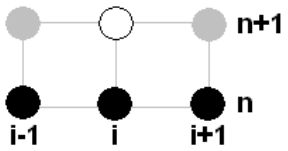

Figure 12: Computational Molecule for Lax-Wendroff Method

Note that if we substitute $\frac{1}{2}(a_{i+1}^n + a_{i-1}^n)$ for a_i^n, we have exactly the Lax method.

3.4.2 Solutions

Since the Lax-Wendroff method is somewhat based on the Lax method, it should compute the exact solution for the trivial case (again note that for $\alpha = 1$ we still have

$a_i^{n+1} = a_{i-1}^n$). Of course this trivial case is the largest Courant factor for the Lax-Wendroff method that is stable (this can be seen through a von Neumann stability analysis). Looking at Figure 13, we see it does.

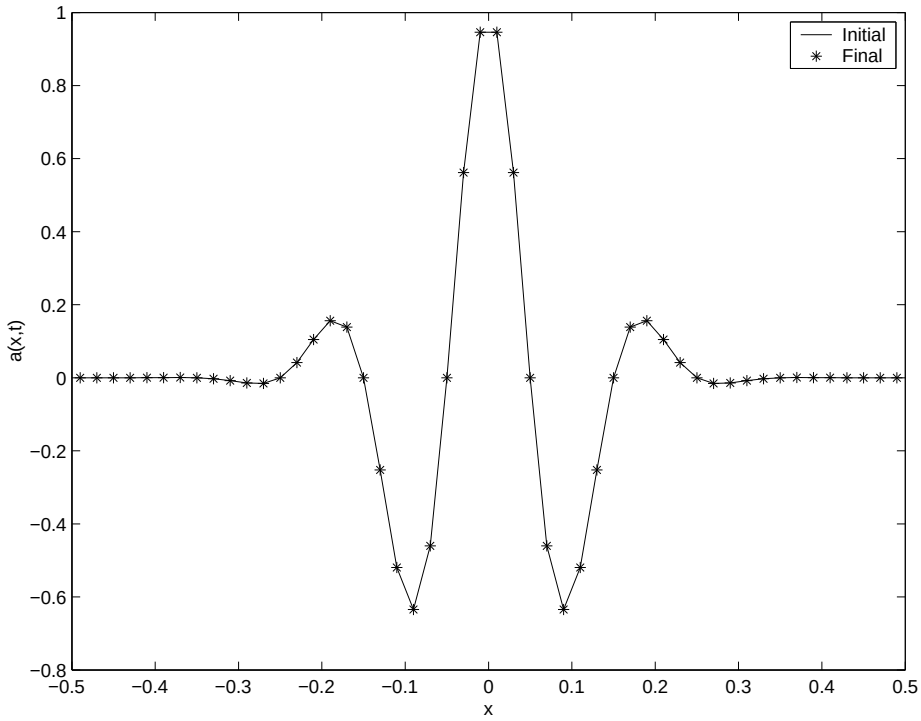

Figure 13: Lax-Wendroff Method for Advection Equation with $\alpha = 1$.

We now want to consider other Courant factors. In Figure 14, we see that this method does not dampen our solution like the previous methods (that is, there is less dissipation), but it does shift the solution slightly with decreasing values of α (there is dispersion).

At the final step, there is actually a wave still present for a Courant factor of 0.1. This is partially because the Lax-Wendroff method has second-order error unlike the Upwind and Lax methods, as stated previously.

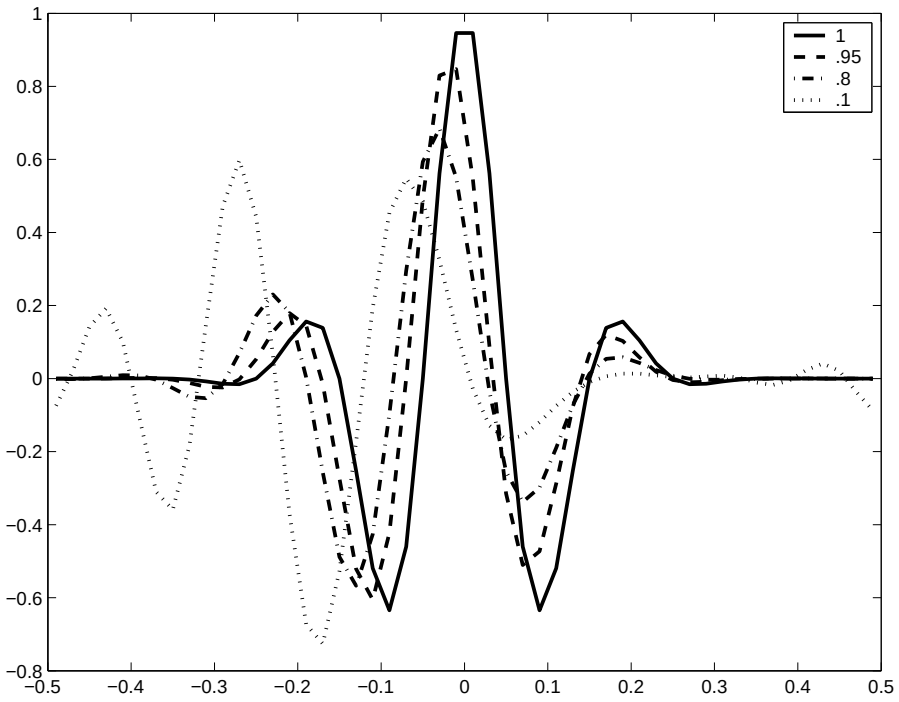

Figure 14: Lax-Wendroff Method for Advection Equation with various α.

3.5 Leapfrog Method

3.5.1 How it works

We saw in previous models that it was better error-wise to use the centered difference form for the derivatives as opposed to the forward or backward differences. For the Leapfrog method, we will replace both derivatives in the advection equation with the corresponding centered difference form - going one step beyond the FTCS method which only used the centered difference for the spatial derivative. Then

$$\frac{\partial a}{\partial t} \approx \frac{a_i^{n+1} - a_i^{n-1}}{2\tau} \text{ and } \frac{\partial a}{\partial x} \approx \frac{a_{i+1}^n - a_{i-1}^n}{2h}.$$

The approximation of the advection equation is then

$$\frac{a_i^{n+1} - a_i^{n-1}}{2\tau} = -c\frac{a_{i+1}^n - a_{i-1}^n}{2h}$$

which implies

$$a_i^{n+1} = a_i^{n-1} - \frac{c\tau}{h}(a_{i+1}^n - a_{i-1}^n).$$

for a graphical version, see Figure 15.

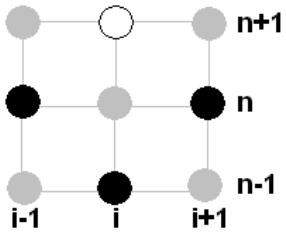

Figure 15: Computational Molecule for Leapfrog Method

It is interesting to note that the Leapfrog method is actually a three-level method. To find the value of the function at one time step, it is necessary to know the value of the function at the previous two time steps. All of our other models have only required the knowledge of one previous time step. As a result, we need initial data at two time levels, or we need to use a two-level scheme to find data at a second time level.

3.5.2 Solutions

Like FTCS, Leapfrog does not have a trivial solution. However, we will use a Courant factor of 1 as a starting point. Looking at Figure 16, we see that the

Leapfrog method reproduces our solution perfectly. Thus we can move on to other Courant factors.

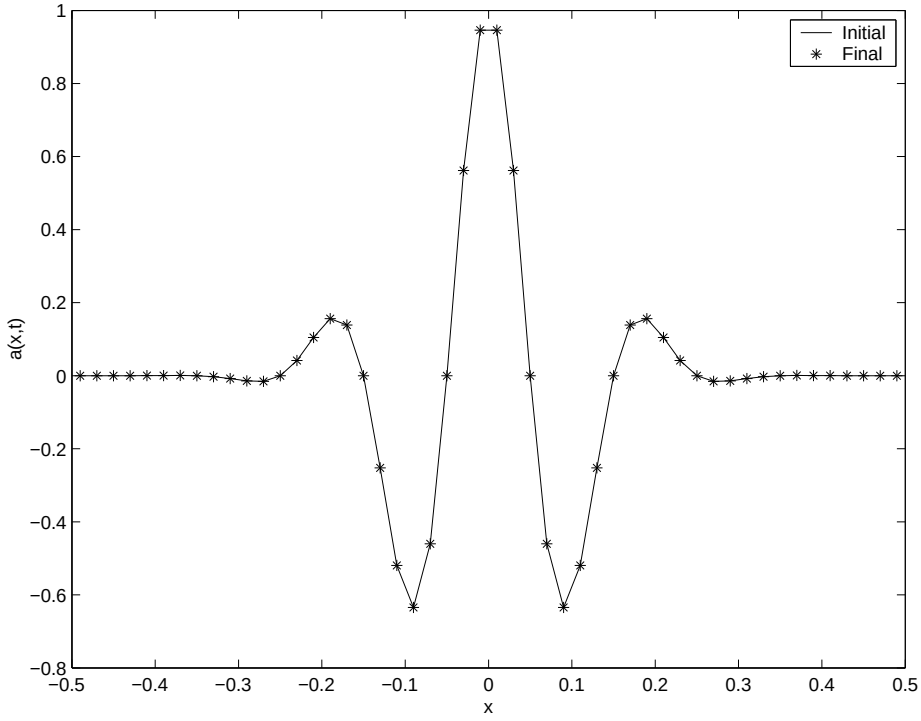

Figure 16: Leapfrog Method for Advection Equation with $\alpha = 1$.

Looking at Figure 17, we see that towards the middle, the Leapfrog method approximates our solution better than the Lax-Wendroff method did. However, on the edges the Leapfrog method overestimates the solution much more than what the Lax-Wendroff method did on the left yet underestimates the solution on the right. There is less dissipation and possibly even less dispersion than in Lax-Wendroff. Thus Leapfrog and Lax-Wendroff are the best so far, but both have different strengths and weaknesses.

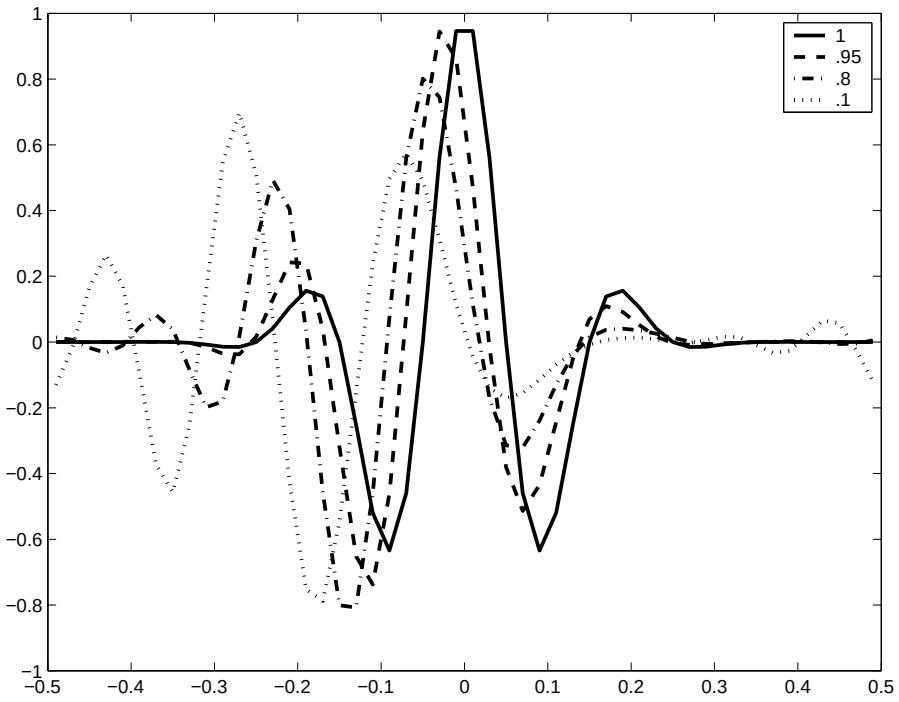

Figure 17: Leapfrog Method for Advection Equation with various α.

3.6 Iterated Crank-Nicholson Method

3.6.1 How it works

For this method, instead of looking at the derivatives computed at a_i^n and plugging them into the advection equation, we will look at the derivatives computed at $a_i^{n+\frac{1}{2}}$. Computing the second-order discretized centered difference formulas, we see

$$\frac{\partial a}{\partial t} \approx \frac{a_i^{n+1} - a_i^n}{\tau}$$

and

$$\frac{\partial a}{\partial x} \approx \frac{1}{2}\left(\frac{\partial a_i^{n+1}}{\partial x} + \frac{\partial a_i^n}{\partial x}\right) \approx \frac{a_{i+1}^{n+1} - a_{i-1}^{n+1} + a_{i+1}^n - a_{i-1}^n}{4h}.$$

So the approximation for the advection equation is

$$a_i^{n+1} - a_i^n = \frac{-c\tau}{4h}(a_{i+1}^{n+1} - a_{i-1}^{n+1} + a_{i+1}^n - a_{i-1}^n)$$

which implies

$$a_i^{n+1} = a_i^n - \frac{c\tau}{4h}(a_{i+1}^{n+1} - a_{i-1}^{n+1} + a_{i+1}^n - a_{i-1}^n).$$

This is the general expression for the Crank-Nicholson method for the advection equation. For a graphical version, see Figure 18.

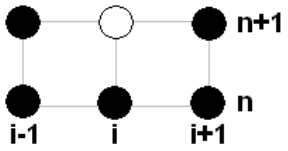

Figure 18: Computational Molecule for Crank-Nicholson Method

Notice that we used the central difference at $(x_i, t_{n+\frac{1}{2}})$, making this method a second-order approximation to the advection equation like the Lax-Wendroff and Leapfrog methods.

Now, compare this difference equation to that given by FTCS:

$$a_i^{n+1} = a_i^n - \frac{c\tau}{2h}(a_{i+1}^n - a_{i-1}^n).$$

Other than a slight difference in the second term on the right hand side, these two equations are very similar. The only difference being that for each value in the second term of FTCS, we average it with its value at the next step. This is true in general for the Crank-Nicholson method.

The Crank-Nicholson method is actually semi-implicit in that it uses values at the $(n+1)$st step to calculate other values at the $(n+1)$st step. One method of solving this is to rewrite it as a matrix equation putting all of the a^{n+1}s on one side:

$$\frac{-c\tau}{4h}a_{i-1}^{n+1} + a_i^{n+1} + \frac{c\tau}{4h}a_{i+1}^{n+1} = \frac{c\tau}{4h}a_{i-1}^{n} - a_i^n - \frac{c\tau}{4h}a_{i+1}^n.$$

So

$$\begin{pmatrix} \ddots & & & \\ \frac{-c\tau}{4h} & 1 & \frac{c\tau}{4h} & 0 \\ 0 & \frac{-c\tau}{4h} & 1 & \frac{c\tau}{4h} \\ & & & \ddots \end{pmatrix} \begin{pmatrix} \vdots \\ a_{i-1}^{n+1} \\ a_i^{n+1} \\ a_{i+1}^{n+1} \\ \vdots \end{pmatrix} = \begin{pmatrix} \ddots & & & \\ \frac{c\tau}{4h} & 1 & \frac{-c\tau}{4h} & 0 \\ 0 & \frac{c\tau}{4h} & 1 & \frac{-c\tau}{4h} \\ & & & \ddots \end{pmatrix} \begin{pmatrix} \vdots \\ a_{i-1}^{n} \\ a_i^{n} \\ a_{i+1}^{n} \\ \vdots \end{pmatrix}.$$

Letting A represent the left-hand side matrix and B the right-hand side matrix, we have $A(a^{n+1}) = B(a^n)$ which implies $(a^{n+1}) = A^{-1}B(a^n)$. Although we can solve this, it is computationally expensive. That is, it takes a large amount of time to compute the solution. Thus, we want to find a different way to solve this method.

An alternative is to use an iterative approach in which we have some sort of initial guess that we can improve upon using Crank-Nicholson. For our purposes, we will use the Lax method for the initial guess, $\tilde{a}_i^{n+1}$ (but any method can work). So for this method, we take a first initial guess

$$\tilde{a}_i^{n+1} = \frac{1}{2}(a_{i+1}^n + a_{i-1}^n) - \frac{c\tau}{2h}(a_{i+1}^n - a_{i-1}^n)$$

then use Crank-Nicholson to improve our guess

$$a_i^{n+1} = a_i^n - \frac{c\tau}{4h}(\tilde{a}_{i+1}^{n+1} - \tilde{a}_{i-1}^{n+1} + a_{i+1}^n - a_{i-1}^n).$$

Of course, if we do this once, we can do it one hundred more times, using Crank-Nicholson to improve guesses made by Crank-Nicholson. When do we stop? Interestingly enough, Teukolsky [T] determined that 2 iterations of Crank-Nicholson is all that is needed to arrive at an appropriate numerical solution. In fact, using more than 2 iterations can cause the solution to be unstable. More specifically, Teukolsky showed that the Iterative Crank-Nicholson method is stable for $4n + 2$ and $4n + 3$ iterations where $n = 0, 1, 2, ...$ Therefore, we will use 2 iterations in all of our solutions.

3.6.2 Solutions

As with every proceeding model, we will begin with $\alpha = 1$. Consider Figures 19-20. It appears that the Crank-Nicholson method slows the wave down.

Crank-Nicholson actually puts a dispersion factor into our solution– one that does not appear in the analytical solution. So for this particular system where analytical solutions are known and simple methods can give exact solutions, Crank-Nicholson is not the ideal method. It is interesting to point out though that, through a von Neumann stability analysis, the Crank-Nicholson method is stable for $\alpha > 1$. In fact, [T] showed that the method is stable for $\alpha \leq 2$. Consider Figure 21.

Notice that this wave with $\alpha = 1.25$ looks quite similar to the waves with smaller Courant factors in Figure 20. The solution does not blow up as it did in the FTCS method at $\frac{1}{5}$ of the crossing time. It is stable because the method is semi-implicit. It uses data at the points to the left and right at the $(n + 1)$st time level as well as

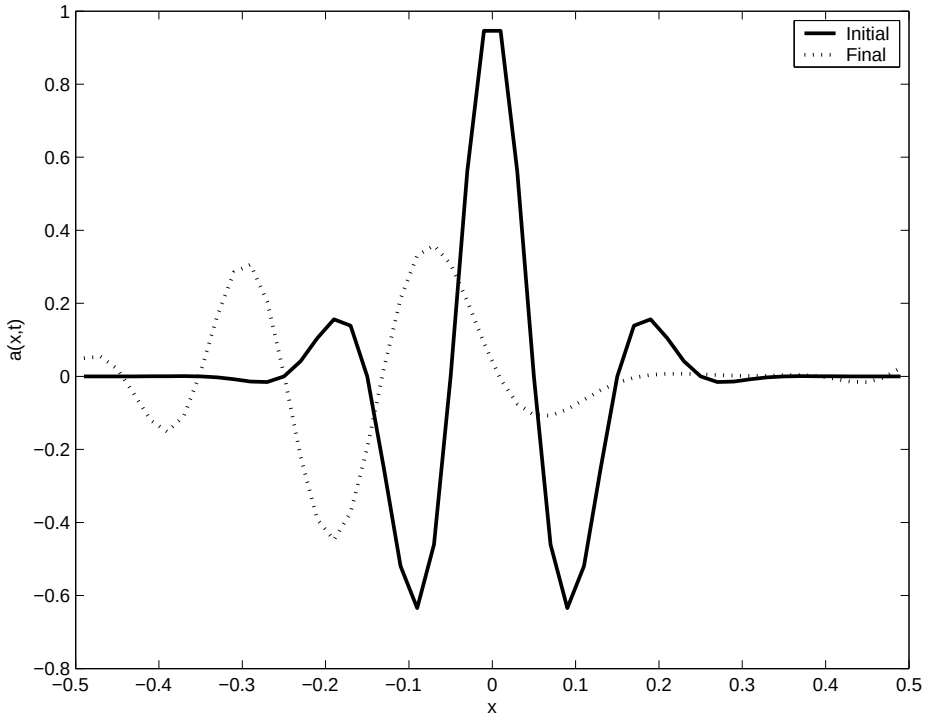

Figure 19: Iterated Crank-Nicholson method for Advection Equation with $\alpha = 1$.

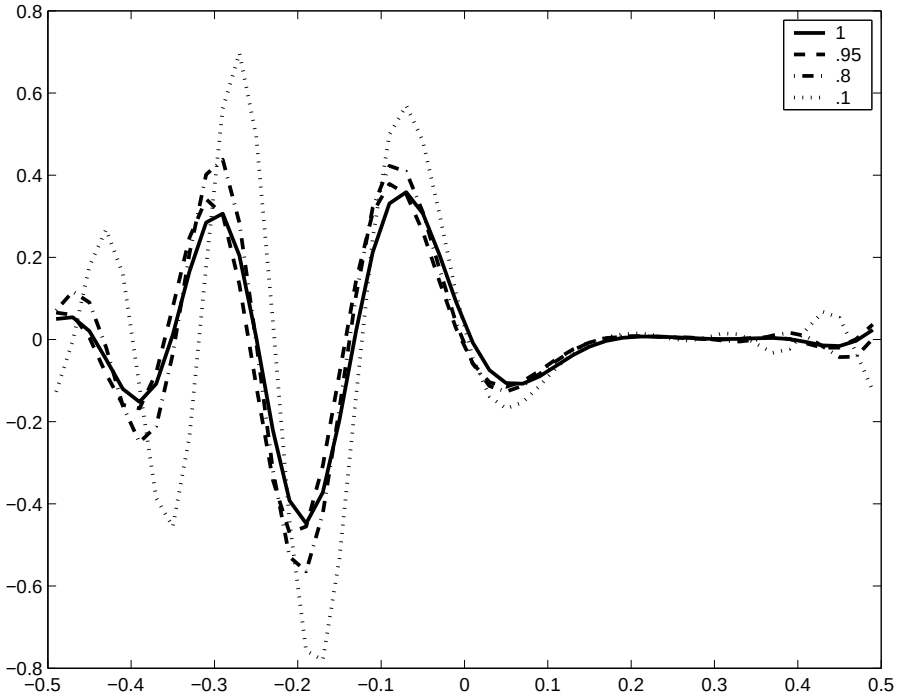

Figure 20: Iterated Crank-Nicholson method for Advection Equation with various α.

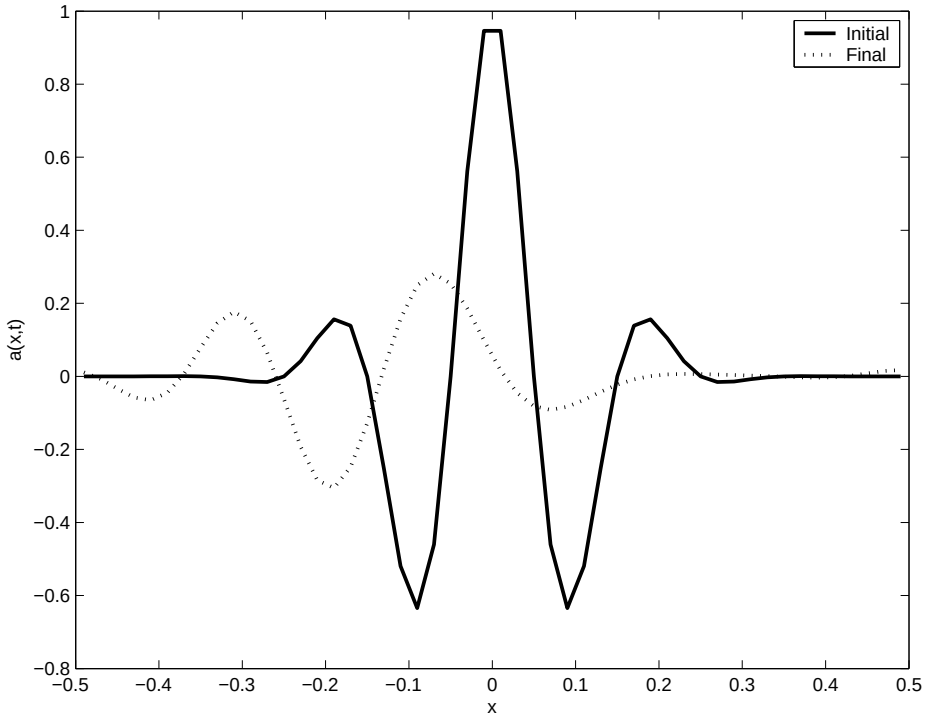

Figure 21: Iterated Crank-Nicholson method for Advection Equation with $\alpha = 1.25$.

the nth level (as we saw in the computational molecule); that helps to prevent the solution from becoming unstable.

4 Wave Equation with Periodic Boundary Conditions

The simple, second-order wave equation in one dimension,

$$u_{tt} = c^2 u_{xx},$$

has general solutions $u(x,t) = a(x+ct) + b(x-ct)$ for arbitrary functions a, b. It generalizes the advection equation to waves moving in two directions as opposed to just one. As with the advection equation, all numerical solutions will be calculated with $c = 1$ and $L = 1$, where L is the length of the grid. Also, we will have 50 grid zones (thus the grid spacing is again $h = .02$) and, unless otherwise noted, we will take the total time for our algorithm to be the time it takes the wave to move one complete grid length, namely $c\tau = L$. Recall from Chapter 3 that this is one crossing time. Thus the only variable is the time step, τ, although we will use $\alpha = \frac{c\tau}{h}$, the Courant factor. We will again use periodic boundary conditions and in the next chapter we will consider other boundary conditions. Similarly, we will solve the equation in the particular case where the initial condition is the cosine-modulated Gaussian pulse,

$$u(x,0) = \cos(kx) \exp\left(\frac{x^2}{2\sigma^2}\right).$$

This time, however, since the wave equation is second-order, we will need another initial condition, this one on $u_t(x,0)$. We will take

$$u_t(x,0) = -cu_x(x,0)$$

so that our wave is right moving and is also a solution to the right-moving advection equation at $t = 0$. This will allow us to see what happens at the boundary points better than if we had the wave going in two directions. In the latter case, we would have waves going off the grid at the same time we had waves reappearing from the other side and reflection would be hard to see.

Since it is often difficult to solve a second-order partial differential equation, we will view the wave equation as a system of first-order equations. There are many different options for this system. We will consider the following two systems. The first is a three-variable system:

$$\begin{cases} u_t = v \\ v_t = c^2 w_x \\ w_t = v_x \end{cases}$$

where $v(x, 0) = -cu_x$ (making our wave right-moving only) and $w(x, 0) = u_x(x, 0)$.

The second is a two-variable system:

$$\begin{cases} v_t = cu_x \\ u_t = cv_x \end{cases}$$

where $v(x, 0) = \frac{1}{c} \int u_t(x, 0)dx$. To make our wave move only to the right, this time we must exploit the fact that $u_t = -cu_x$ and $u_t = cv_x$ to get $v_x = -u_x$. Since we have the freedom to choose our initial condition, we will set $v(x, 0) = -u$. We will prove later that these equations are equivalent to the wave equation.

4.1 Three-Variable System

We must first believe that the three-variable system is equivalent to the wave equation.

Proposition 4.1. *For some functions $u(x,t), v(x,t), w(x,t)$, the partial differential equation $u_{tt} = c^2 u_{xx}$ is equivalent to the system of equations*

$$\left|\begin{array}{l} u_t = v \\ v_t = c^2 w_x \\ w_t = v_x \end{array}\right.$$

where $w(x,0) = u_x(x,0)$.

Proof. First, assume $u_{tt} = c^2 u_{xx}$ for some function u. Set $v = u_t$ and $w = u_x$. Then

$$v_t = u_{tt} = c^2 u_{xx} = c^2 w_x$$

and

$$w_t = u_{xt} = u_{tx} = v_x.$$

Thus, a solution of the wave equation leads to a solution of the three-variable system.

Now assume $u_t = v$, $v_t = c^2 w_x$, and $w_t = v_x$, with initial condition $w(x,0) = u_x(x,0)$. It is easy to see

$$u_{tt} = v_t = c^2 w_x = c^2 u_{xx}$$

at time $t = 0$. Now, we need to show $w = u_x$ for all t, that is, $\frac{\partial}{\partial t}(w - u_x) = 0$. We know $\frac{\partial}{\partial t} w = v_x = u_{tx}$. Hence

$$0 = u_{tx} - u_{xt} = \frac{\partial}{\partial t} w - \frac{\partial}{\partial t} u_x = \frac{\partial}{\partial t}(w - u_x)$$

which implies $w = u_x$ for all t. Thus, the wave equation has a solution. Therefore, the wave equation and the system are equivalent. □

4.1.1 Lax method

As before, the Lax method simply takes the equations for FTCS and replaces the first term with the average of the neighboring points. Thus, our equations are

$$u_i^{n+1} = .5(u_{i+1}^n + u_{i-1}^n) + \tau v_i^n,$$

$$v_i^{n+1} = .5(v_{i+1}^n + v_{i-1}^n) + \tau c^2 \left(\frac{w_{i+1}^n - w_{i-1}^n}{2h} \right),$$

$$w_i^{n+1} = .5(w_{i+1}^n + w_{i-1}^n) + \tau \left(\frac{v_{i+1}^n - v_{i-1}^n}{2h} \right).$$

Now, we want to look at our solution. Consider Figure 22(a).

This graph displays the wave for various Courant factors after one crossing time. Recall from the Lax method in Chapter 3 that for a Courant factor of 1, our numerical solution at any point is the same as our initial wave (except for a shift in the axis as our wave moves in time). The wave had exactly the same amplitude as the initial wave. Thus at one crossing time, we want all solutions to look like that of $\alpha = 1$. In our picture however, the amplitude of the wave decreases with decreasing Courant factor. We saw this with the advection equation and noted that it was dissipation.

Notice that all of our solutions have been computed for a fairly small amount of time – 1 crossing time. It is possible that the error might build up in time, making our method unstable. To make sure this does not happen, we want to look at what

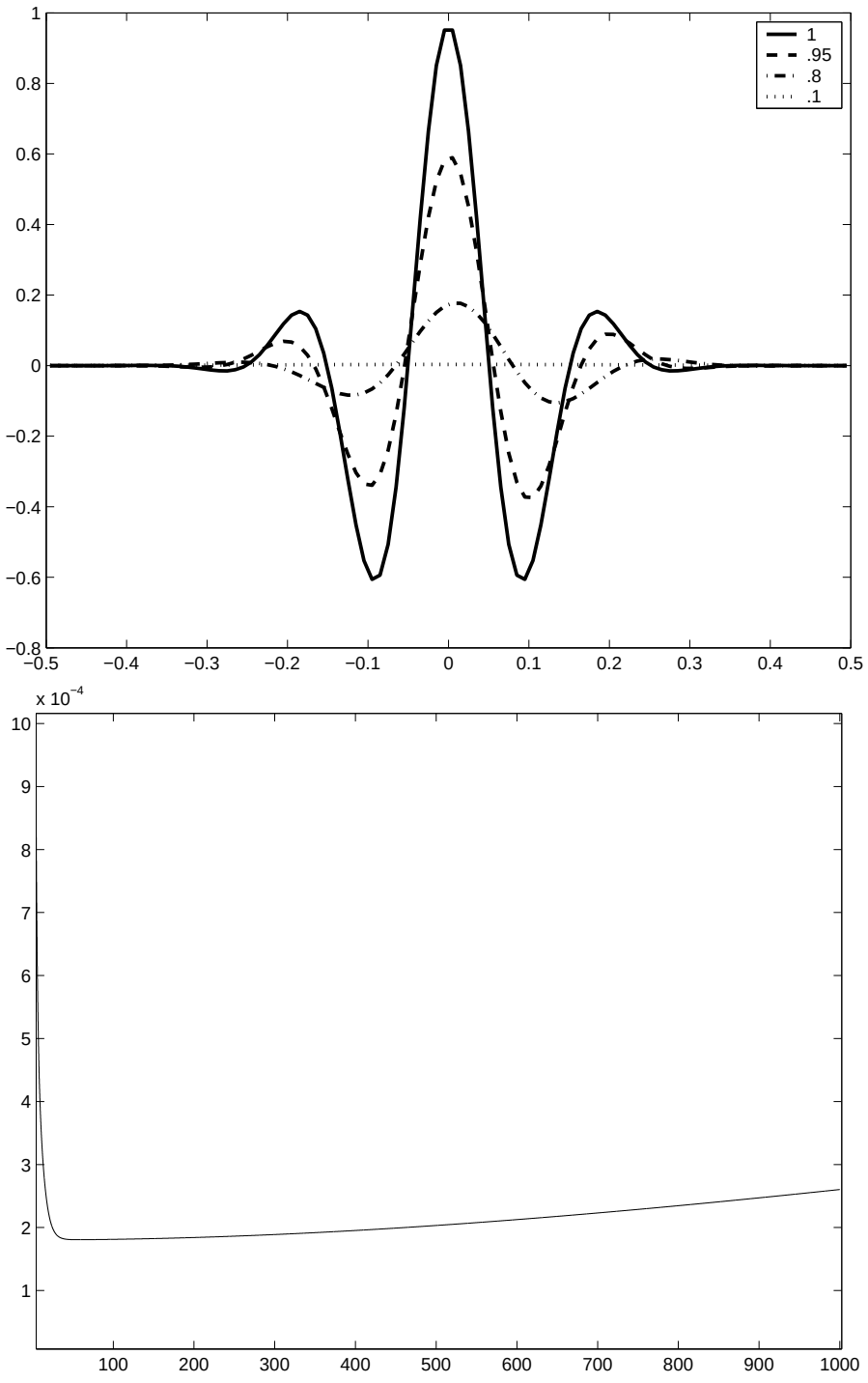

Figure 22: Lax method for the Three-Variable System Wave Equation with periodic B.C. Plot (a) is for one crossing time and various α and plot (b) is a plot of the norm for $\alpha = .8$.

happens to the wave over a long period of time. Consider Figure 22(b), where we compute the norm of u at each time step up to 1000 crossing times.

Being that the magnitude is on the order of 10^{-4}, we can definitely say there is dissipation in the Lax method because the norm decreases during the first few crossing times. Interestingly enough, over a long period of time the norm begins to increase but this is a result of error, most likely from the boundary conditions. The important thing to notice from this plot, however, is that the magnitude of u is constant. The Lax method does not blow up, at least up to 1000 crossing times. This is great; however as we saw, the Lax method still gives us dissipation, so we will look at our next method.

4.1.2 Lax-Wendroff method

Recall that Lax-Wendroff adds a term to the equations from FTCS by finding the second-order Taylor Series of the wave. Since $u_{tt} = c^2 u_{xx}$, $v_{tt} = c^2 w_{xt} = c^2 v_{xx}$, and $w_{tt} = v_{xt} = c^2 w_{xx}$, Lax-Wendroff gives us

$$u_i^{n+1} = u_i^n + \tau v_i^n + \frac{\tau^2 c^2}{2}\left(\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2}\right),$$

$$v_i^{n+1} = v_i^n + \tau c^2\left(\frac{w_{i+1}^n - w_{i-1}^n}{2h}\right) + \frac{\tau^2 c^2}{2}\left(\frac{v_{i+1}^n - 2v_i^n + v_{i-1}^n}{h^2}\right),$$

$$w_i^{n+1} = w_i^n + \tau\left(\frac{v_{i+1}^n - v_{i-1}^n}{2h}\right) + \frac{\tau^2 c^2}{2}\left(\frac{w_{i+1}^n - 2w_i^n + w_{i-1}^n}{h^2}\right).$$

Now we shall examine our numerical solution. Consider Figure 23(a). Unlike the Lax method, there is not nearly as much dissipation (though there is a small amount as the amplitude of the wave becomes slightly smaller with decreasing

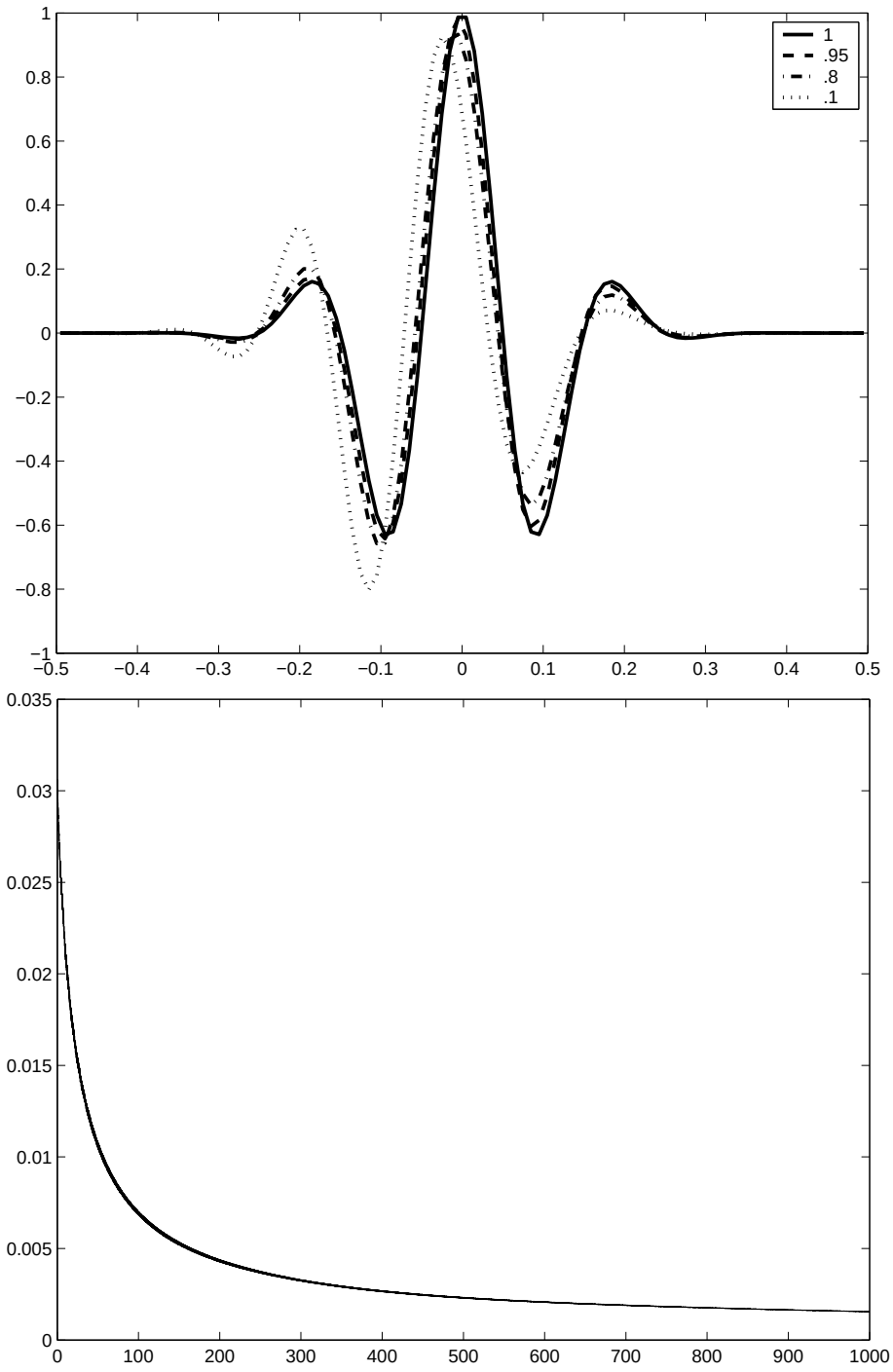

Figure 23: Lax-Wendroff method for the Three-Variable System Wave Equation with periodic B.C. Plot (a) is for one crossing time and various α and plot (b) is a plot of the norm for $\alpha = .8$.

Courant factor). Notice, however, that for a Courant factor of .1, the wave is shifted a little to the left. Lax-Wendroff does introduce some dispersion, which we do not want. For $\alpha \approx 1$, though, Lax-Wendroff gives us a close approximation.

Before moving on to our next method, let us again consider the norm of u over time, just to make sure our solution is stable. Consider Figure 23(b). Although Lax-Wendroff does peak at the beginning, over time it decreases (that is, it dissipates) and eventually approaches a constant value on the order of 10^{-3}. This is similar to what we saw in the Lax method and what we want to see all along. We do not want the solution to blow up.

4.1.3 Leapfrog method

For this case, we will actually use a different grid to get our difference equations. Our lattice will be constructed so that we find the values of u at the lattice points (i, n), the values of v at the lattice points $(i, n+\frac{1}{2})$, and the values of w at the lattice points $(i+\frac{1}{2}, n)$. Consider the computational molecule in Figure 24 to get a better understanding of the grid points and which variable is evaluated at each point.

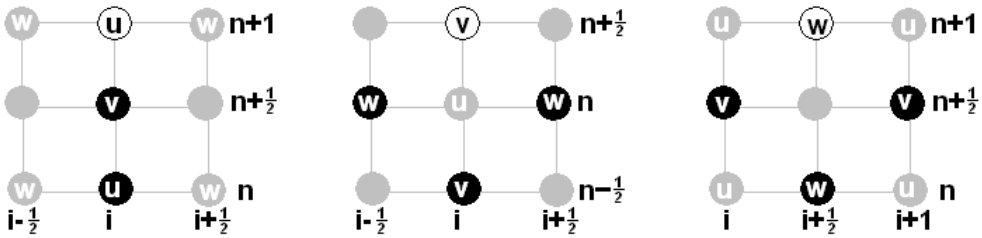

Figure 24: Computational Molecule for the Three-Variable Leapfrog

One could argue that this is the most natural method of computing second-

order derivatives accurately in this system. Using this approach, we need only to find boundary points for v as w is not on the boundary and u can be determined from the values of v (since $u_t = v$). Our difference equations are then

$$v_i^{n+\frac{1}{2}} = v_i^{n-\frac{1}{2}} + \tau c^2 \left(\frac{w_{i+\frac{1}{2}}^n - w_{i-\frac{1}{2}}^n}{h} \right),$$

$$w_{i+\frac{1}{2}}^{n+1} = w_{i+\frac{1}{2}}^n + \tau \left(\frac{v_{i+1}^{n+\frac{1}{2}} - v_{i-1}^{n+\frac{1}{2}}}{h} \right),$$

$$u_i^{n+1} = u_i^n + \tau v_i^{n+\frac{1}{2}}.$$

Of course, now that we are evaluating v on half steps of τ, we must analyze our initial value of v. In previous methods, we simply calculated the initial value at $\tau = 0$. Thus, we have two options: we can numerically integrate to the $\frac{\tau}{2}$ half step or calculate the analytical value of v at $\frac{\tau}{2}$ by plugging $x - \frac{c\tau}{2}$ into our initial equation. It turns out that numerically integrating introduces more error than if we simply calculated what the initial value would be. Thus, we will analytically calculate our initial value of v at $(i, \frac{\tau}{2})$. Recall from our derivation of this system that $w = u_x$. Thus our initial value from w is $w_{i+\frac{1}{2}}^0 = \frac{1}{h}(u_{i+1}^0 - u_i^0)$ (notice that we could also solve this analytically, but interestingly enough, the difference does not affect our solutions).

Recall from our previous chapter that while Leapfrog has dispersion like Lax-Wendroff, the Leapfrog method had less dissipation. Let us see what happens for the wave equation. Consider Figure 25(a).

This is by far our best method yet. Although we can see a bit of dispersion

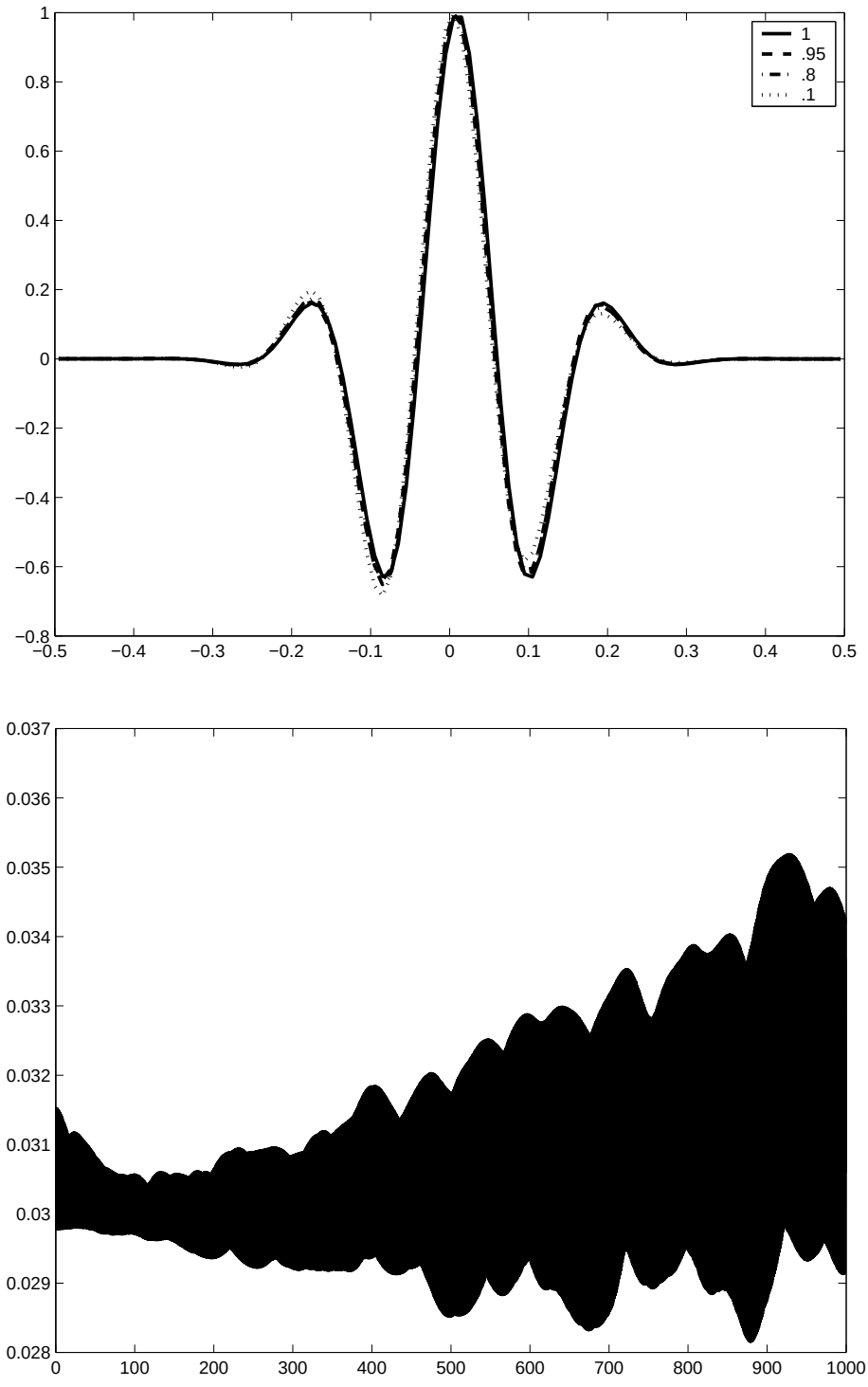

Figure 25: Leapfrog method for the Three-Variable System Wave Equation with periodic B.C. Plot (a) is for one crossing time and various α and plot (b) is a plot of the norm for $\alpha = .8$.

starting to build up for smaller Courant factors (notice the amplitude is slightly large on the right for a Courant factor of .1), this method approximates our solution beautifully.

Let us consider the magnitude of u, to make sure the method does not blow up over time. Figure 25(b), although a bit more jagged than our previous methods, shows us exactly what we want to see – the error remains fairly constant over time. The method (up to 1000 crossing times) is stable. Notice that the magnitude of u is much larger than in our previous methods. This is because the Leapfrog method does not introduce nearly as much dissipation as our previous methods did. Also, the solution is, on average, growing in magnitude, very slowly, over time.

4.1.4 Iterated Crank-Nicholson method

Recall from before that for the Crank-Nicholson method, we took each term in the second term on the right hand side of the FTCS method and replaced it by the average of that value and the value of the function at the next time step. Since the FTCS method would give us

$$u_i^{n+1} = u_i^n + \tau v_i^n,$$

$$v_i^{n+1} = v_i^n + \tau c^2 \left(\frac{w_{i+1}^n - w_{i-1}^n}{2h} \right),$$

$$w_i^{n+1} = w_i^n + \tau \left(\frac{v_{i+1}^n - v_{i-1}^n}{2h} \right),$$

the iterated Crank-Nicholson method would give us the following:

$$u_i^{n+1} = u_i^n + \frac{\tau}{2}(v_i^{n+1} + v_i^n),$$

$$v_i^{n+1} = v_i^n + \frac{\tau c^2}{2}\left(\frac{w_{i+1}^{n+1} - w_{i-1}^{n+1} + w_{i+1}^n - w_{i-1}^n}{2h}\right),$$
$$w_i^{n+1} = w_i^n + \frac{\tau}{2}\left(\frac{v_{i+1}^{n+1} - v_{i-1}^{n+1} + v_{i+1}^n - v_{i-1}^n}{2h}\right).$$

As we saw with the advection equation, the Crank-Nicholson method actually added a dispersion factor. It causes different wavelengths in the wave to travel at different speeds. It turns out that the same thing happens with the wave equation, as we see in Figure 26(a).

The Crank-Nicholson method adds a large amount of dispersion that the other methods do not. Like Leapfrog however, there does not seem to be dissipation among Courant factors. In fact, one could guess that the opposite happens from our other methods, that is, dissipation occurs for *larger* Courant factors. However, this is not dissipation and is more likely a result of dispersion. We see this because the left half of the wave has a smaller amplitude for smaller Courant factors while the right half of the wave has a larger amplitude.

The Crank-Nicholson method does not work as well as Leapfrog did for the wave equation, but let us consider the magnitude of u over time anyway. Figure 26(b) shows that the Crank-Nicholson method is actually quite similar to Leapfrog over time. The magnitude is much larger (again due to lack of dissipation) and is rather jagged. Of course, it is constant – which is the important thing. It does not blow up.

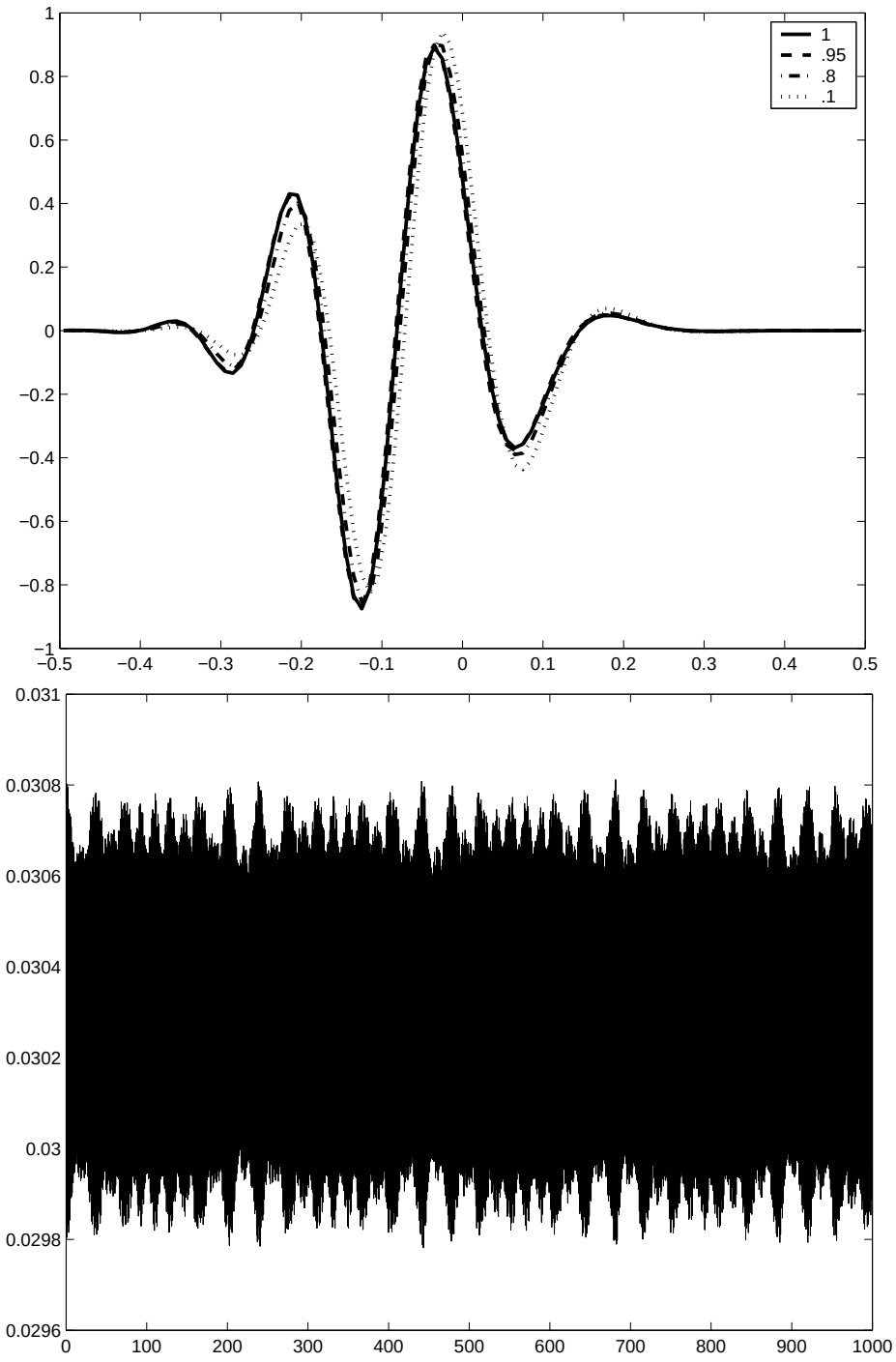

Figure 26: Crank-Nicholson method for the Three-Variable System Wave Equation with periodic B.C. Plot (a) is for one crossing time and various α and plot (b) is a plot of the norm for $\alpha = .8$.

4.2 Two-Variable System

As with the three-variable system, we must first believe that the two-variable system is equivalent to the wave equation.

Proposition 4.2. *For some functions $u(x,t), v(x,t)$, the partial differential equation $u_{tt} = c^2 u_{xx}$ is equivalent to the system of equations*

$$\begin{cases} v_t = cu_x \\ u_t = cv_x \end{cases}$$

Proof. If $u_t = cv_x$ and $v_t = cu_x$ for functions u, v, then $u_{tt} = cv_{xt} = cv_{tx} = c^2 u_{xx}$.

Now assume $u_{tt} = c^2 u_{xx}$ for some function u. There exists a family of functions v such that $v_t = cu_x$. Then each v has the form $v(x,t) = \int cu_x dt + f(x)$ for some function f. Now, we choose f so that $cv_x = u_t$ at $t = 0$. Therefore, we need only to show that $cv_x = u_t$ always, that is, $\frac{\partial}{\partial t}(cv_x - u_t) = 0$. Since $\frac{\partial}{\partial t} cv_x = cv_{xt} = cv_{tx} = c^2 u_{xx}$, we have $\frac{\partial}{\partial t}(cv_x - u_t) = c^2 u_{xx} - u_{tt} = 0$. Thus the wave equation and the system are equivalent. □

4.2.1 Lax method

As we saw earlier, the Lax method yields the following set of equations:

$$u_i^{n+1} = .5(u_{i+1}^n + u_{i-1}^n) + \tau c \left(\frac{v_{i+1}^n - v_{i-1}^n}{2h} \right),$$

$$v_i^{n+1} = .5(v_{i-1}^n + v_{i-1}^n) + \tau c \left(\frac{u_{i+1}^n - u_{i-1}^n}{2h} \right).$$

Figure 27 shows the numerical data.

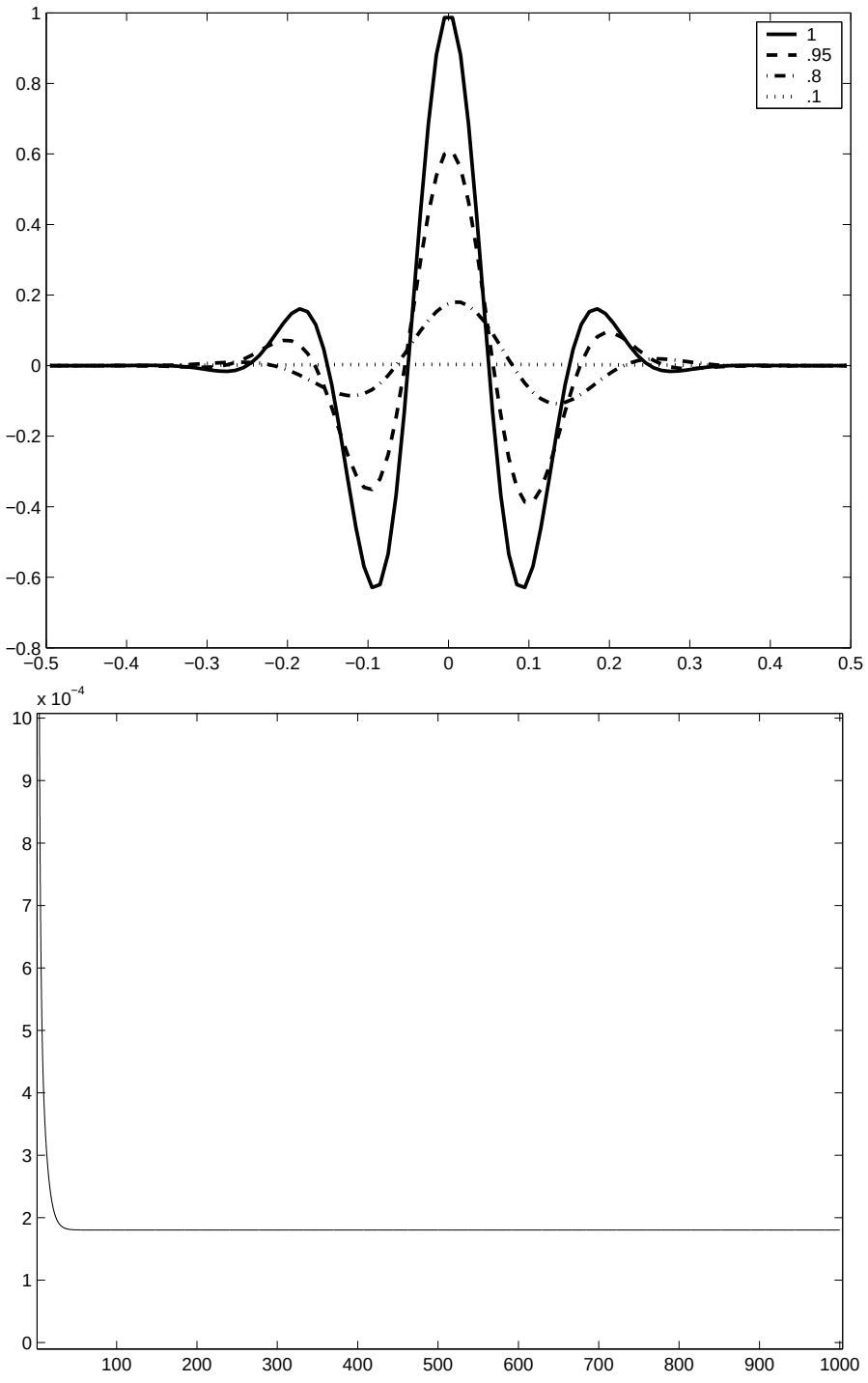

Figure 27: Lax method for the Two-Variable System Wave Equation with periodic B.C. Plot (a) is for one crossing time and various α and plot (b) is a plot of the norm for $\alpha = .8$.

This graph is almost identical to that of the Lax method in the three-variable system, Figure 22(a). We still see severe dissipation, but luckily no dispersion. Looking at our plot of the norm of u, Figure 27(b), we again see a similarity to the three-variable system. The only difference being that with the three-variable system, we saw a slight increase in the magnitude, but here it looks quite constant.

4.2.2 Lax-Wendroff method

Since $u_{tt} = c^2 u_{xx}$ and $v_{tt} = cw_{xt} = c^2 v_{xx}$, Lax-Wendroff gives us

$$u_i^{n+1} = u_i^n + \tau c \left(\frac{v_{i+1}^n - v_{i-1}^n}{2h} \right) + \frac{\tau^2 c^2}{2} \left(\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h} \right),$$

$$v_i^{n+1} = v_i^n + \tau c \left(\frac{u_{i+1}^n - u_{i-1}^n}{2h} \right) + \frac{\tau^2 c^2}{2} \left(\frac{v_{i+1}^n - 2v_i^n + v_{i-1}^n}{h} \right).$$

Now, we want to look at the solution. Consider Figures 28(a) and (b).

We again see a similarity between the two- and three-variable system. Notice the slight difference in the magnitude of dispersion between Figure 28(a) and Figure 23(a). At the position -0.1 on the x-axis, we see there is a slightly larger amount of dispersion for the two-variable case. This would leave us to believe that the three-variable system approximates the wave equation better. However, we have not compared the norm plots, Figures 28(b) and 23(b). Although Lax-Wendroff does peak at the beginning, over time it decreases (that is, it dissipates) and eventually approaches a constant value on the order of 10^{-3}. This is similar to what we saw for the three-variable case and what we want to see all along. We do not want the solution to blow up.

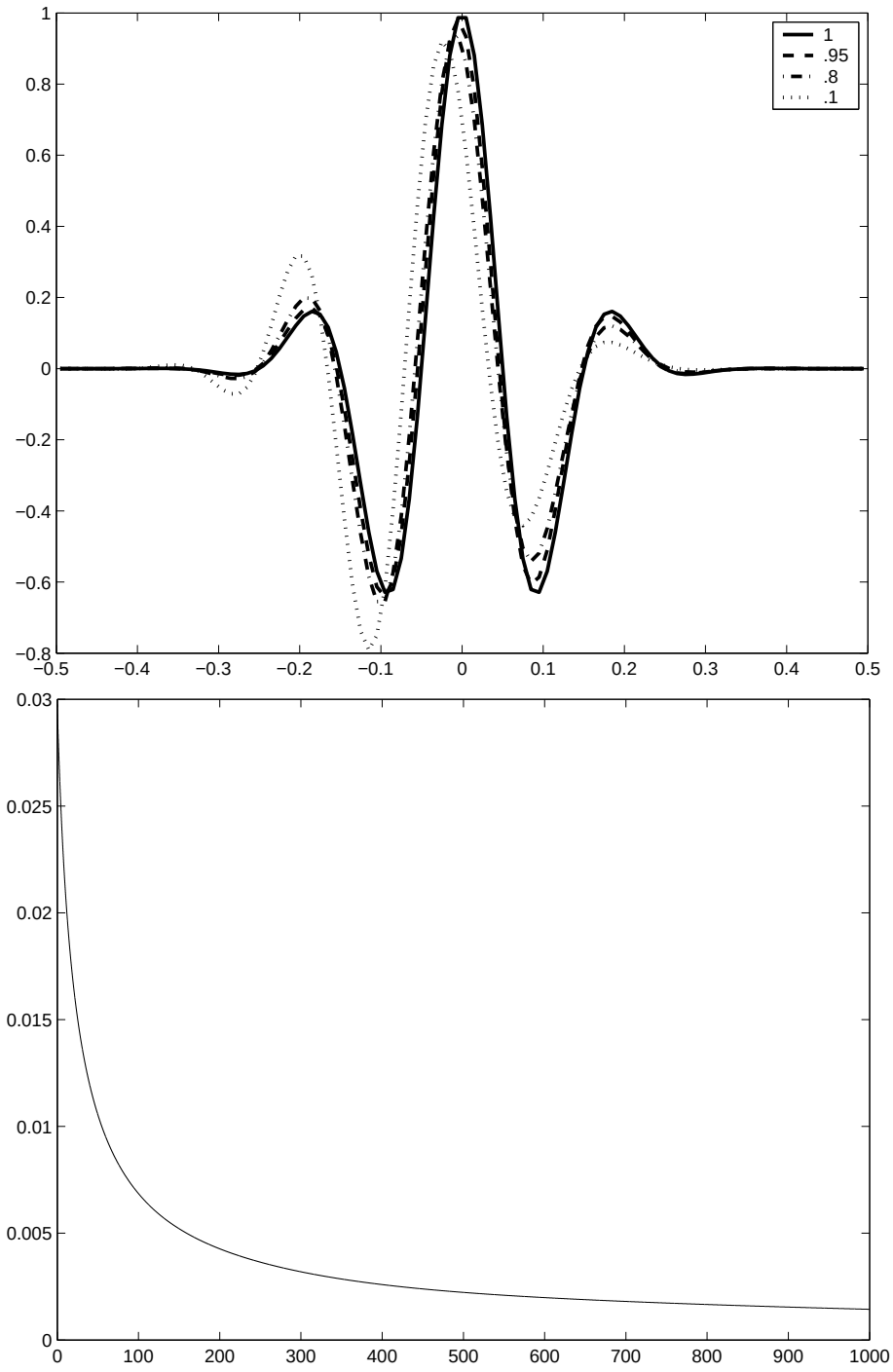

Figure 28: Lax-Wendroff method for the Two-Variable System Wave Equation with periodic B.C. Plot (a) is for one crossing time and various α and plot (b) is a plot of the norm for $\alpha = .8$.

4.2.3 Leapfrog method

Our difference equations for Leapfrog are

$$u_i^{n+1} = u_i^n + c\tau \left(\frac{v_{i+\frac{1}{2}}^{n+\frac{1}{2}} - v_{i-\frac{1}{2}}^{n+\frac{1}{2}}}{h} \right),$$

$$v_{i+\frac{1}{2}}^{n+\frac{1}{2}} = v_{i+\frac{1}{2}}^{n-\frac{1}{2}} + c\tau \left(\frac{u_{i+1}^n - u_i^n}{h} \right).$$

Consider the computational molecule in Figure 29 to get a better understanding of the grid points and which variable is evaluated at each point.

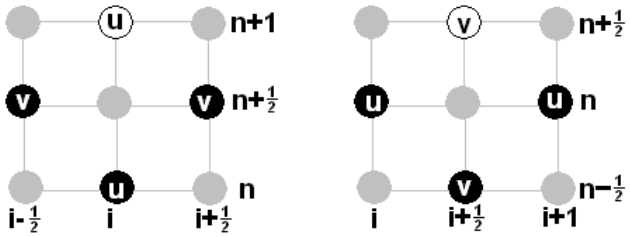

Figure 29: Computational Molecule for the Two-Variable Leapfrog

Again, we want to look at the numerical solutions. Consider Figure 30(a) in comparison with Figure 25(a).

We see in our two-variable system the same wave as in the three-variable system. Both solutions approximate the wave solution quite well with no dissipation and very little dispersion. Let us look at a norm plot. Figure 30(b) shows us practically the same amount of error as the three-variable system did, with one exception. Although they are both quite jagged, we can see that the two-variable system does not increase as the three-variable system did. However, the increase was so slight and was over

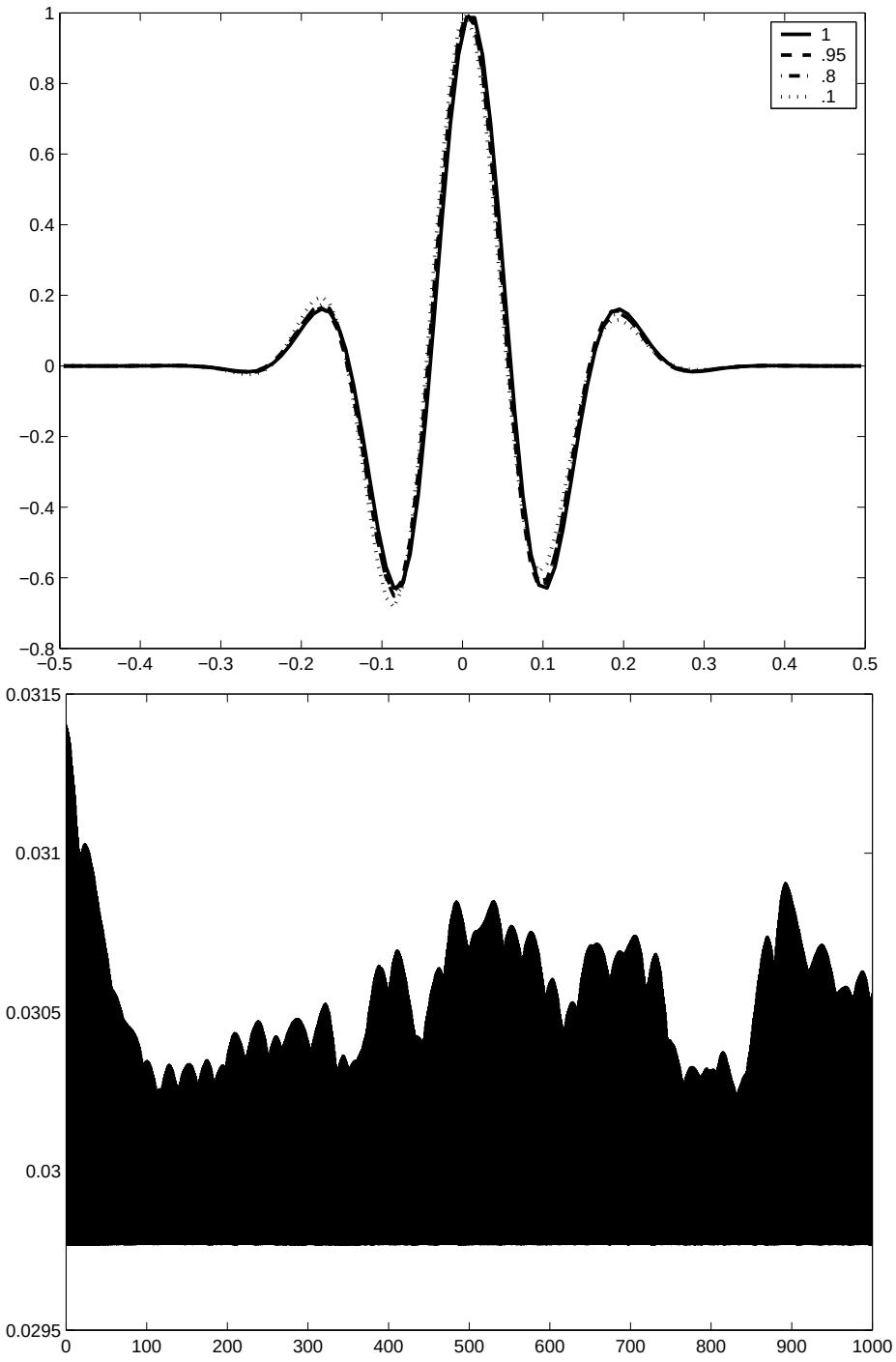

Figure 30: Leapfrog method for the Two-Variable System Wave Equation with periodic B.C. Plot (a) is for one crossing time and various α and plot (b) is a plot of the norm for $\alpha = .8$.

such a long matter of time that this really does not tell us anything. Both methods are stable, though, which is important.

4.2.4 Iterated Crank-Nicholson method

The equations for the iterated Crank-Nicholson method are

$$u_i^{n+1} = u_i^n + \frac{c\tau}{2}\left(\frac{v_{i+1}^{n+1} - v_{i-1}^{n+1} + v_{i+1}^n - v_{i-1}^n}{2h}\right),$$

$$v_i^{n+1} = v_i^n + \frac{c\tau}{2}\left(\frac{u_{i+1}^{n+1} - u_{i-1}^{n+1} + u_{i+1}^n - u_{i-1}^n}{2h}\right).$$

In the three-variable system approximation to the wave equation, we saw that the Crank-Nicholson method, though it had little dissipation, introduced a large amount of dispersion after one crossing time and blew up by 100 crossing times. We want to see if the two-variable system gives us the same. Consider Figures 31(a) and (b).

As with our other three methods, the two cases are quite similar. The magnitude of the norm of u is slightly smaller for the two-variable system, but not enough to amount to anything.

4.3 Summary

In looking at the numerous graphs between the different methods and between the two- and three-variable systems, there are many things to notice. First, the Lax method, though it gave us no dispersion, had severe dissipation. All three other methods, however, gave us little to no dissipation. We will thus discontinue our

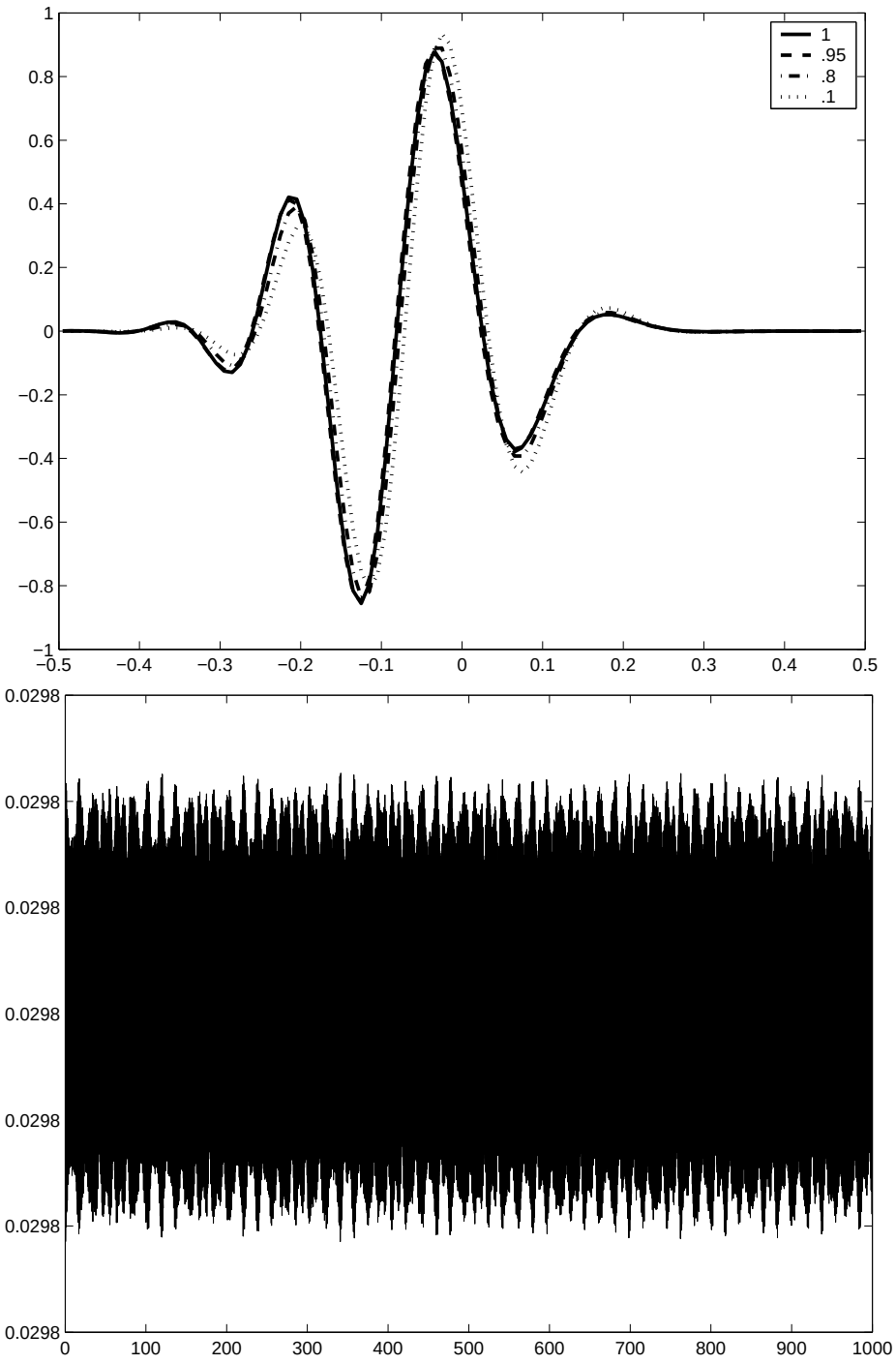

Figure 31: Crank-Nicholson method for the Two-Variable System Wave Equation with periodic B.C. Plot (a) is for one crossing time and various α and plot (b) is a plot of the norm for $\alpha = .8$.

study of the Lax method in favor of the less dissipative second-order methods. Also, we notice that the Leapfrog method is the best approximation between the remaining three as it has an extremely small amount of dispersion.

Beyond the methods though, there are differences between the two- and three-variable wave systems. Although there was not a large difference between the two, we saw that over longer periods of time, the two-variable system actually approximated our solution better. Also, recall that it is more simplistic and computationally faster to calculate with two variables as opposed to three. Plus, the variables in the two-variable system are completely analogous (except for in Leapfrog where v is defined on different grid points).

In the next chapter, we will look at both the two-variable and three-variable systems and focus on various boundary conditions, other than periodic.

5 Other Boundary Conditions

We have been using periodic boundary conditions so far to estimate our solution and accuracies of integration methods, but what if we use different conditions? We want to use boundary conditions that will not cause reflection at the boundaries. Below, we consider two other types of boundary conditions: outgoing and Dirichlet.

5.1 Dirichlet Boundary Conditions

Dirichlet boundary conditions give specific values at the boundaries. In our case, we will put the analytical value of the solution at the boundary points. Thus it is important to note that this only works for equations for which the solution is known. In other cases, we can impose a known approximate solution.

5.1.1 Three-Variable System

Lax-Wendroff method Since we are using outgoing boundary conditions, all of our wave will have left the grid by one crossing time. Therefore, any remaining wave in our numerical solution will be error in our method. This error can be attributed to the boundary values (perhaps reflection occurs) or the initial conditions (perhaps a residual wave is left over after the first time step that does not move off the grid). Instead of moving a potentially erroneous solution off the grid, Dirichlet boundary conditions give the exact analytical value of the equation at the boundaries. If our numerical solution is not equivalent to the analytical solution, then Dirichlet

boundary conditions will create a discontinuity in the solution at the boundaries. This could lead to reflections. Consider Figures 32-33.

In plot (a), we see that our wave still has a magnitude of approximately .015, which is small, but not negligible. Perhaps more interesting, however, is the comparison between plots (a) and (b). Since plot (b) is at 1.3 crossing times, if this remaining error were due to reflections at the right-hand boundary, we would expect our wave to have moved to the left. However, it has not. This tells us that this particular wave is not moving, and being that it is in the middle, has not moved since the initial time step. Thus, our error is actually a residual wave from the initial data. Notice that this residual wave does not depend on the Courant factor. The Courant factor really only comes in on the right hand side of plot(a), where we see some reflections at the boundary. Regardless of the amplitude of the wave near the boundaries, Dirichlet boundary conditions impose what can be a significantly different value at the boundary. Since the amplitude of the wave at $i = 1$ and $i = 2$ may be very different, we end up with reflections at the boundaries. However, we notice that this reflection is quite small for the Lax-Wendroff method. Looking at Figure 32, we see that the reflection is on the order of 10^{-4}. Let us consider the norm plot, Figure 33. Dirichlet boundary conditions give us a constant value for the norm of u. This tells us that the method is stable, which is exactly what we want. It is what we expect, though, because the boundary condition is constantly resetting values at the boundary to a particular value – it does not let the numerical value of the wave change so drastically. So, even though we see reflections, we will

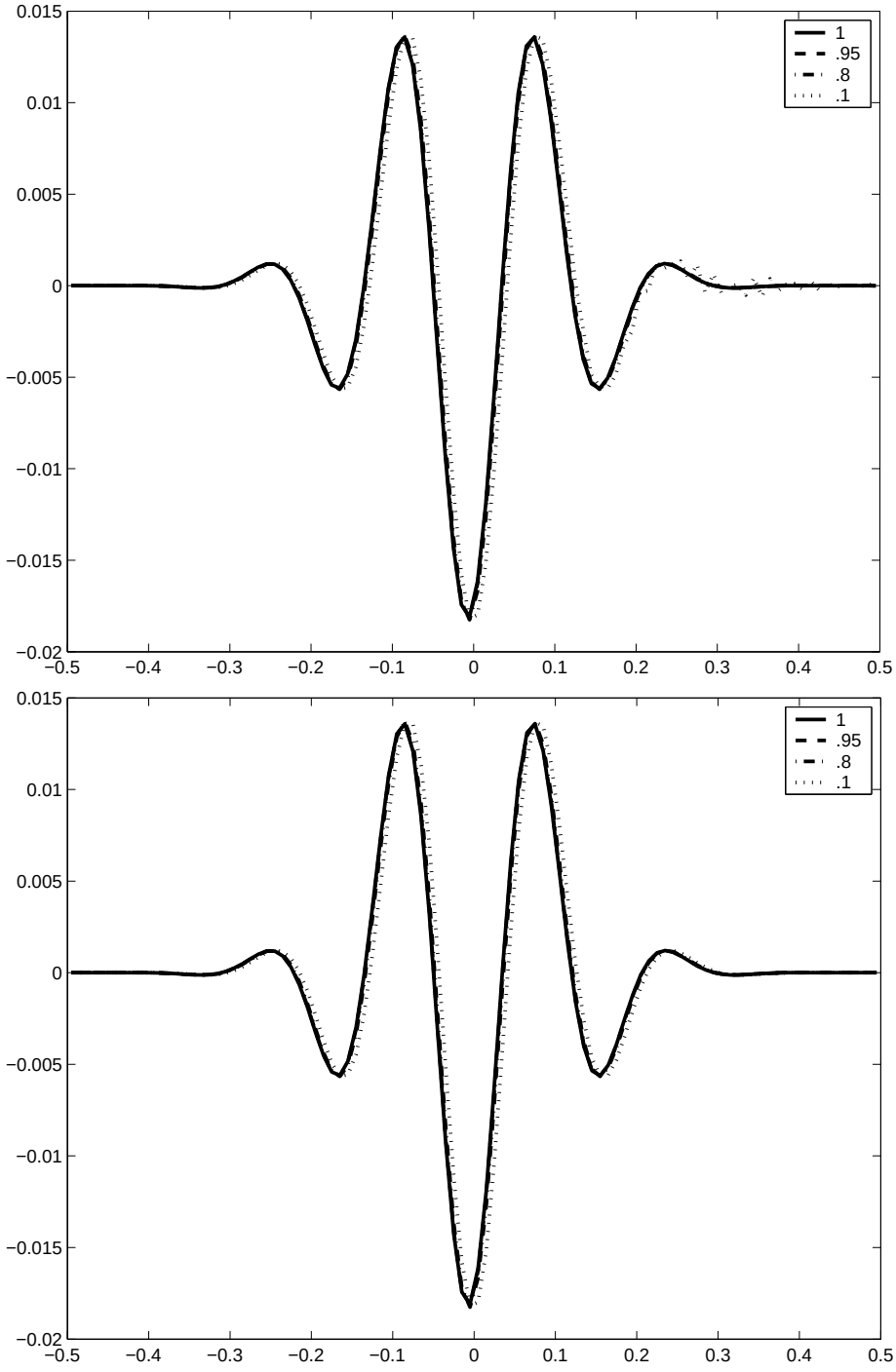

Figure 32: Lax-Wendroff method for Three-Variable Wave Equation with Dirichlet B.C. (a) One crossing time and various α; (b) 1.3 crossing times and various α.

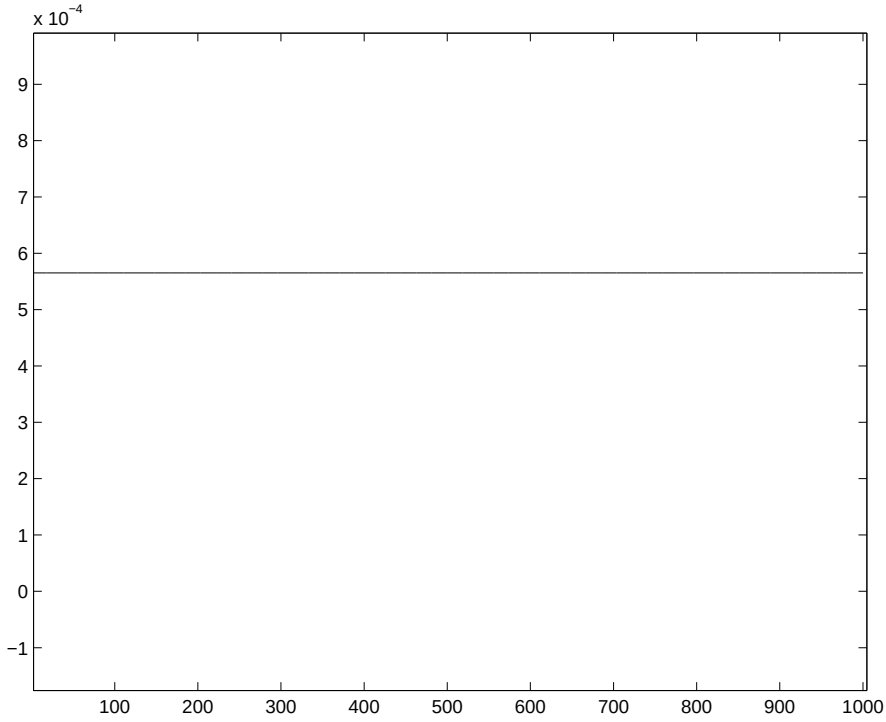

Figure 33: Norm plot for Lax-Wendroff method for Three-Variable Wave Equation with Dirichlet B.C. for $\alpha = .8$.

continue to look at Dirichlet boundary conditions as a possibility for solving a more complex system of equations in the next chapter.

Leapfrog method Consider Figures 34-35.

Notice that this time we do not appear to have a residual wave. However, since the error is greater for Leapfrog by a factor of 10 there may be a residual wave but it is too small to appear on the plot. As we stated with Lax-Wendroff, the wave should be completely off the grid by one crossing time; however, we still have a wave. Thus we have a large amount of reflection unlike in Lax-Wendroff. Looking at the norm plot, Figure 35, we see that our value for the norm of u is still constant and so Leapfrog is a stable method.

Iterated Crank-Nicholson method Recall that the equations are

$$u_i^{n+1} = u_i^n + \frac{\tau}{2}(v_i^{n+1} + v_i^n),$$

$$v_i^{n+1} = v_i^n + \frac{\tau c^2}{2}\left(\frac{w_{i+1}^{n+1} - w_{i-1}^{n+1} + w_{i+1}^n - w_{i-1}^n}{2h}\right),$$

$$w_i^{n+1} = w_i^n + \frac{\tau}{2}\left(\frac{v_{i+1}^{n+1} - v_{i-1}^{n+1} + v_{i+1}^n - v_{i-1}^n}{2h}\right).$$

For the numerical results, consider Figures 36-37.

The plots of Crank-Nicholson are quite similar to those of Leapfrog. We see that the magnitude of the error is slightly higher for crossing times of 1 and 1.3 and that we cannot tell whether there is a residual wave (but following the pattern, there most likely is). The norm plots have a smaller range of values than in the periodic

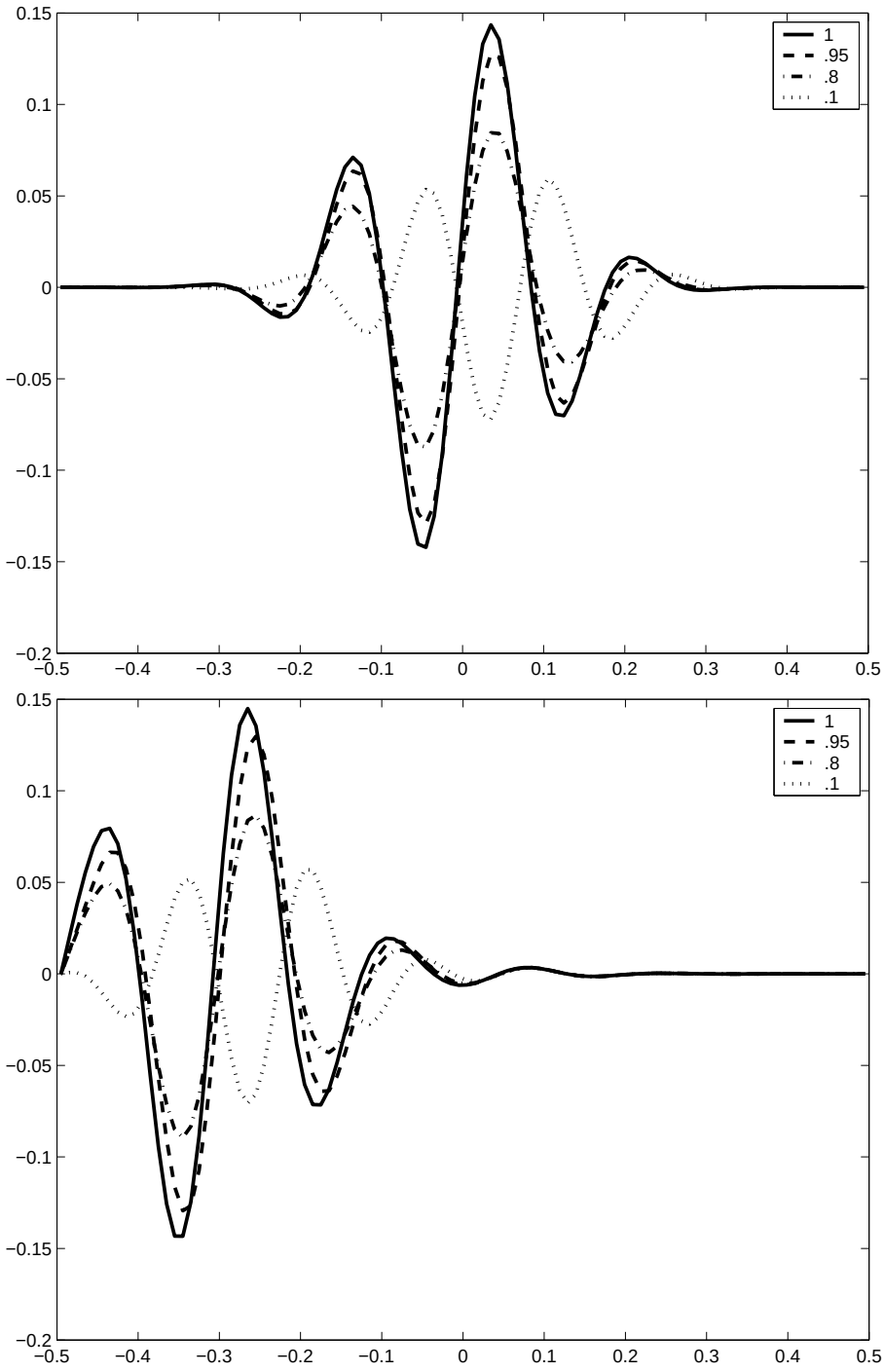

Figure 34: Leapfrog method for Three-Variable Wave Equation with Dirichlet B.C. (a) One crossing time and various α; (b) 1.3 crossing times and various α.; Error Plot for $\alpha = .8$.

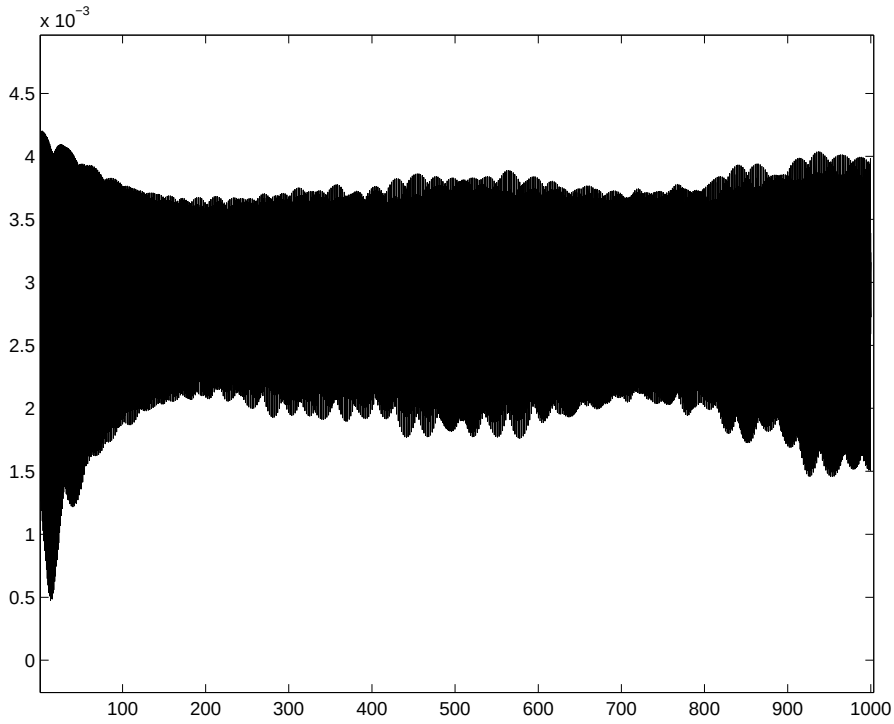

Figure 35: Norm plot for Leapfrog method for Three-Variable Wave Equation with Dirichlet B.C. for $\alpha = .8$.

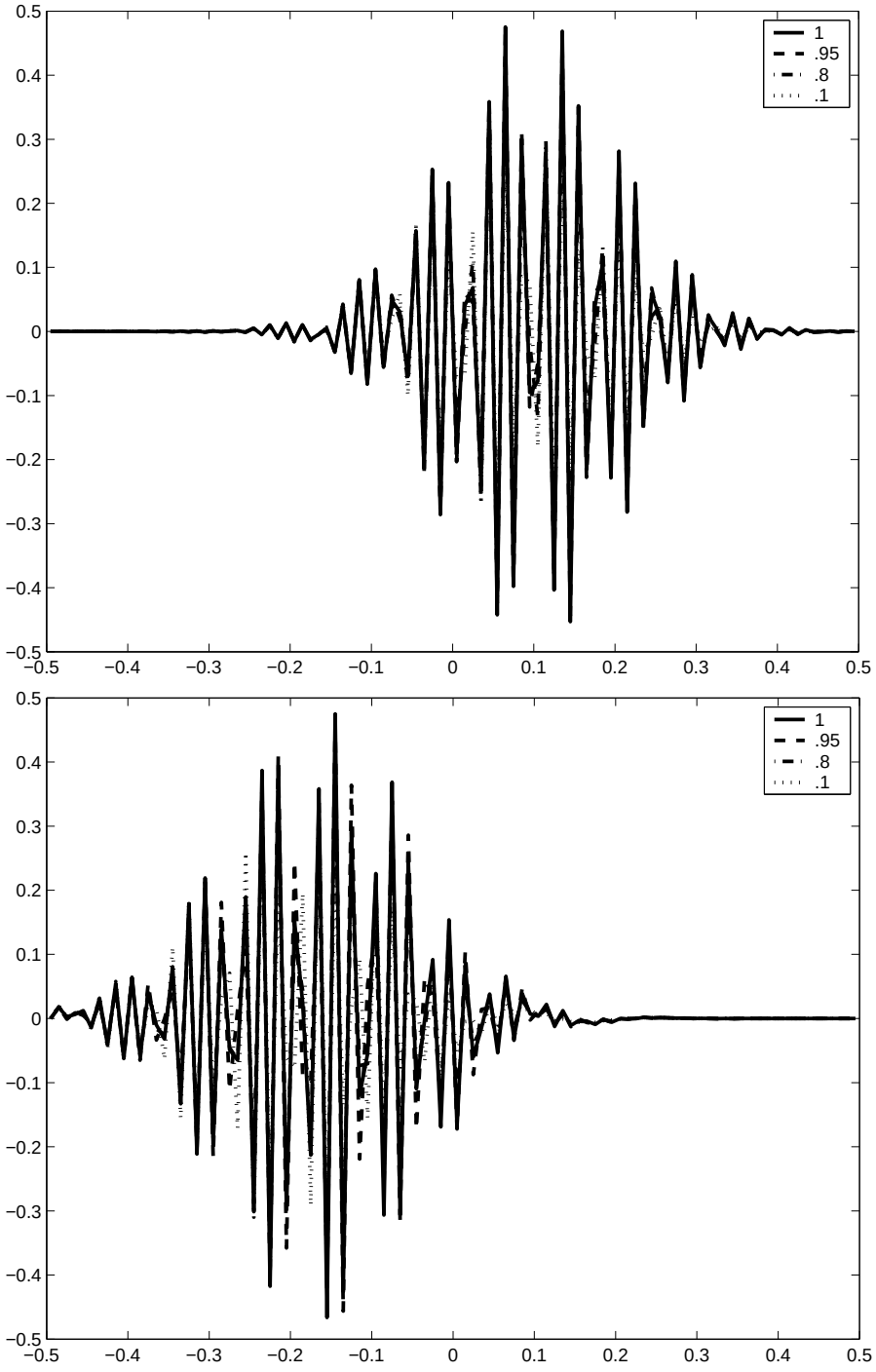

Figure 36: Crank-Nicholson method for Three-Variable Wave Equation with Dirichlet B.C. (a) One crossing time and various α; (b) 1.3 crossing times and various α.

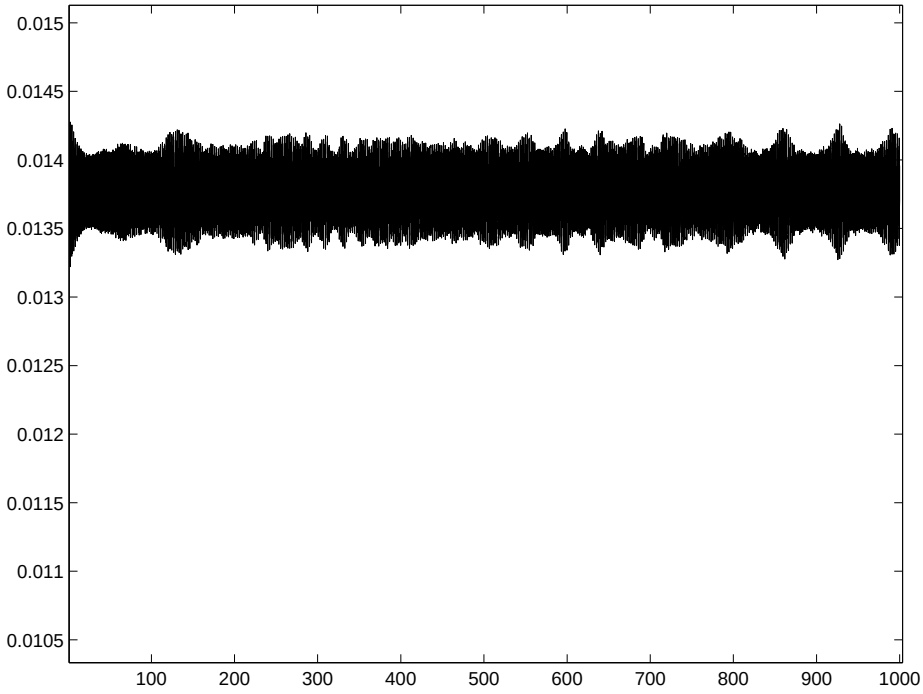

Figure 37: Norm plot for Crank-Nicholson method for Three-Variable Wave Equation with Dirichlet B.C. for $\alpha = .8$.

case, but this is expected as the Dirichlet boundary condition sets particular values at the boundaries instead of letting them grow with the rest of the solution.

5.1.2 Two-Variable System

Lax-Wendroff method Let's examine what happens in Figure 38.

What we want to see in plot (a) is a wave with zero value across the board. However, we have a wave with amplitude up to 15. We also see that the wave is approximately the same for all Courant factors. In the three-variable system, only the residual wave for different Courant factors was the same. However, in plot (b) we see that there is no residual wave and, in fact, it turns out that there will never be a residual wave in the two-variable system. There is definitely a large amount of reflection at the boundaries. This is far worse than what we expect and therefore requires future work to understand the cause. Note that the wave has only reached the boundary once at 1 crossing time. Once each part of the wave reaches the boundary again, it will be given a value of exactly 0 (as the analytical solution is 0). Thus, we would expect the wave, after an initial amount of time where reflection occurs, to go to zero. Looking at the norm plot (Figure 39), we see that after an initial peak in the data, the magnitude of u remains constant at 10^{-4} This is exactly what we expect, as in the three-variable case.

Leapfrog method Again, we want to look at the numerical solutions in Figure 40.

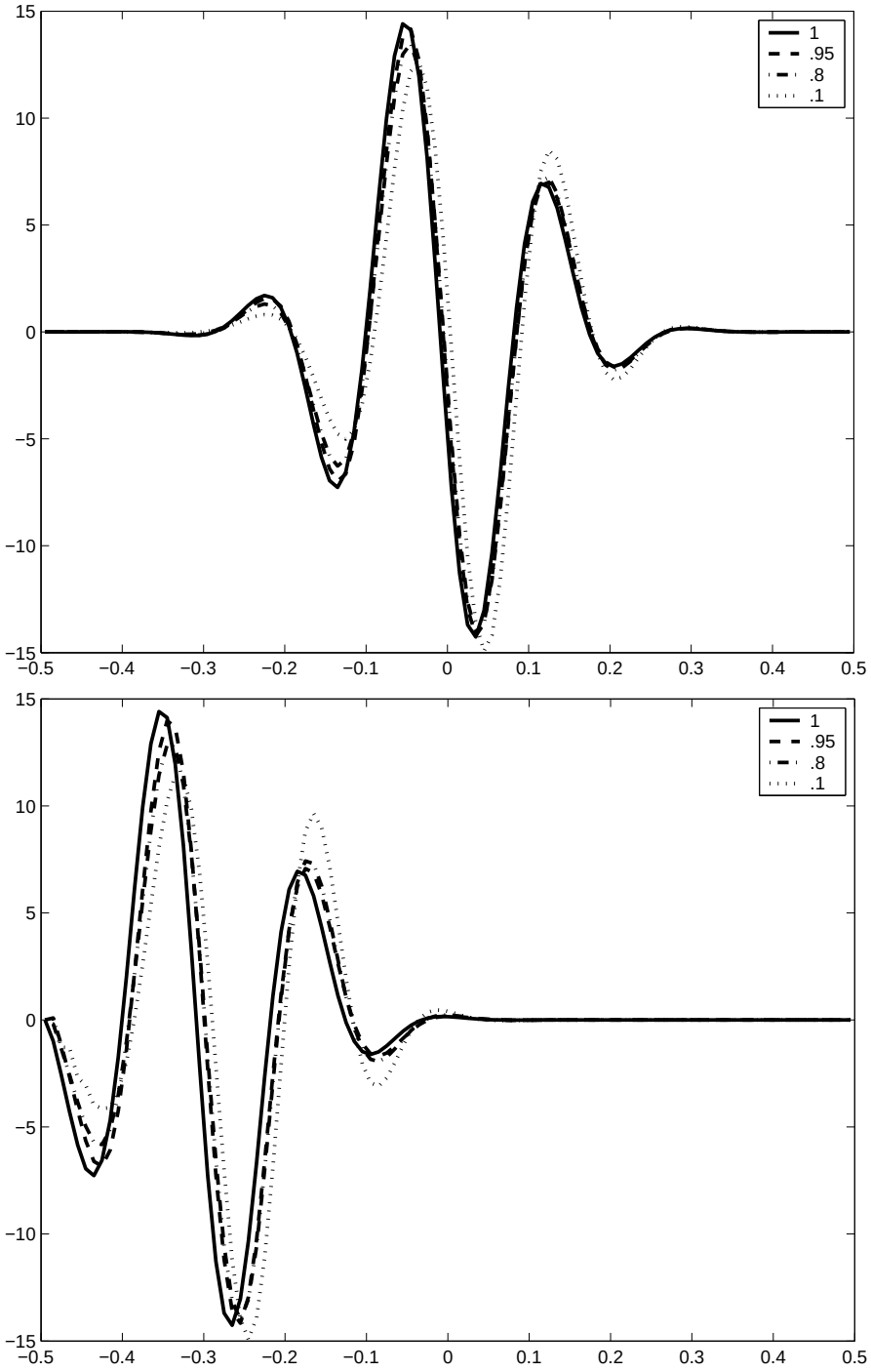

Figure 38: Lax-Wendroff method for Two-Variable Wave Equation with Dirichlet B.C. (a) One crossing time and various α; (b) 1.3 crossing times and various α.

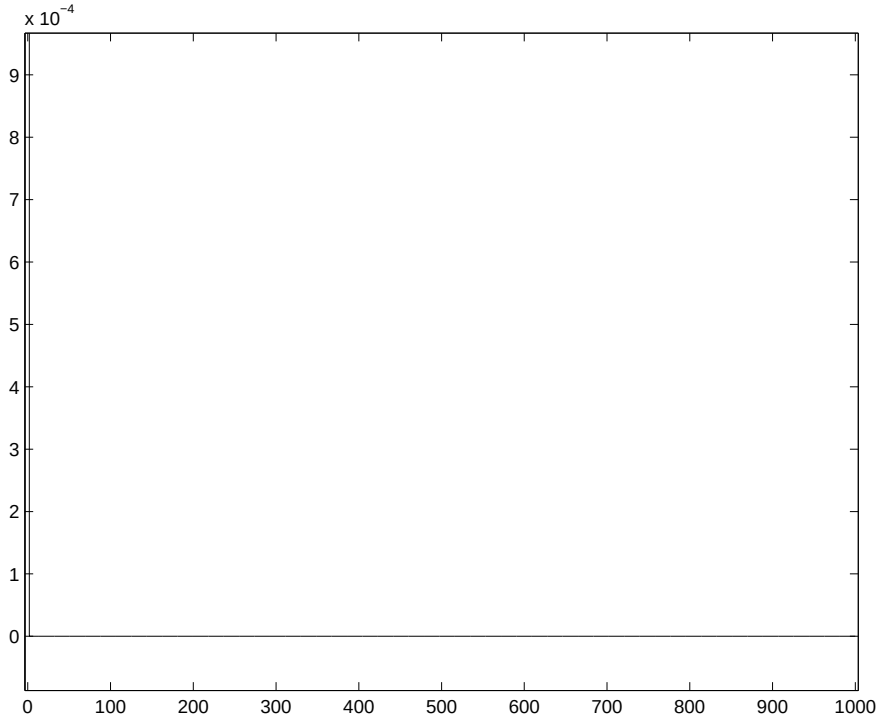

Figure 39: Norm plot for Lax-Wendroff method for Two-Variable Wave Equation with Dirichlet B.C. for $\alpha = .8$.

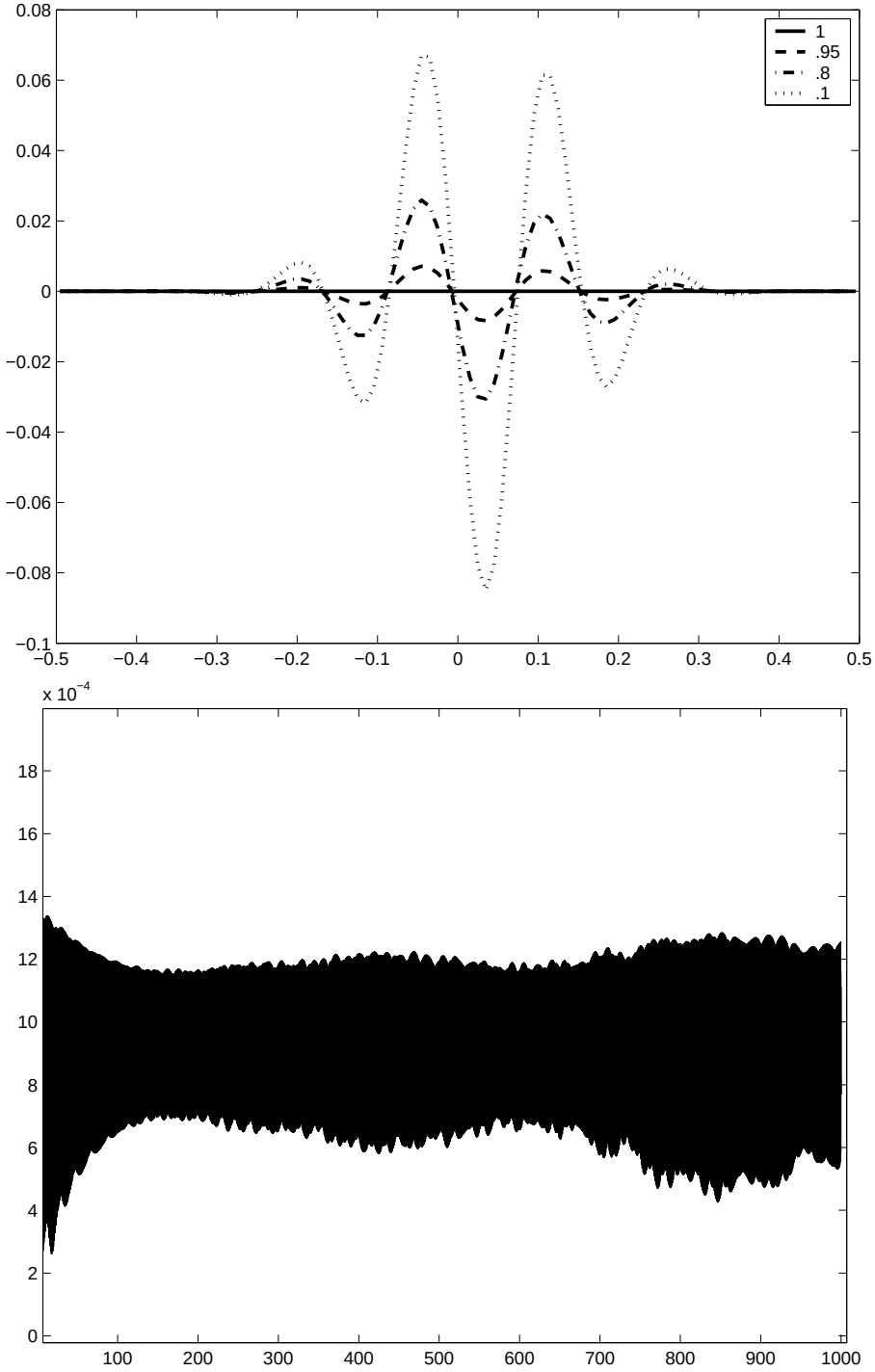

Figure 40: Leapfrog method for Two-Variable Wave Equation with Dirichlet B.C. (a) One crossing time and various α.; (b) Norm Plot for $\alpha = .8$.

Plot (a) looks rather different from Lax-Wendroff. We immediately see the amplitude of the wave increases with decreasing Courant factor, but the amplitude is still smaller than that of Lax-Wendroff. Looking at the norm plot (plot(b)), we still see a constant value for u, which is what we want.

Iterated Crank-Nicholson method Recall with previous boundary conditions that Crank-Nicholson had a tendency to blow up in time. Generally, the graphs seemed to produce garbage. Let us see if we get the same effects with Dirichlet boundary conditions. Consider Figure 41.

Plot (a) actually shows a bit of a wave, but the amplitude is larger than in Leapfrog and Lax-Wendroff. Looking at the norm plot in plot(b), we see again that our method is constant. However for the two-variable case, the magnitude of u is much larger. This is perhaps due to reflection, or error.

5.2 Outgoing Boundary Conditions

The Box method is very similar to the Upwind method, but also uses the point u_{i-1}^{n+1} (for a right-traveling wave). At a boundary point, we know that the wave moves to the left or the right depending on whether it is at the left or right boundary point (otherwise, waves would appear from off the grid). We can use the advection equation (since it is uni-directional) as an approximation for the wave equation at the boundaries. Thus, at the left boundary points, we have a left-traveling wave and so $u_t = cu_x$. We know the second-order derivatives at $u_{i+\frac{1}{2}}^{n+\frac{1}{2}}$ are approximated

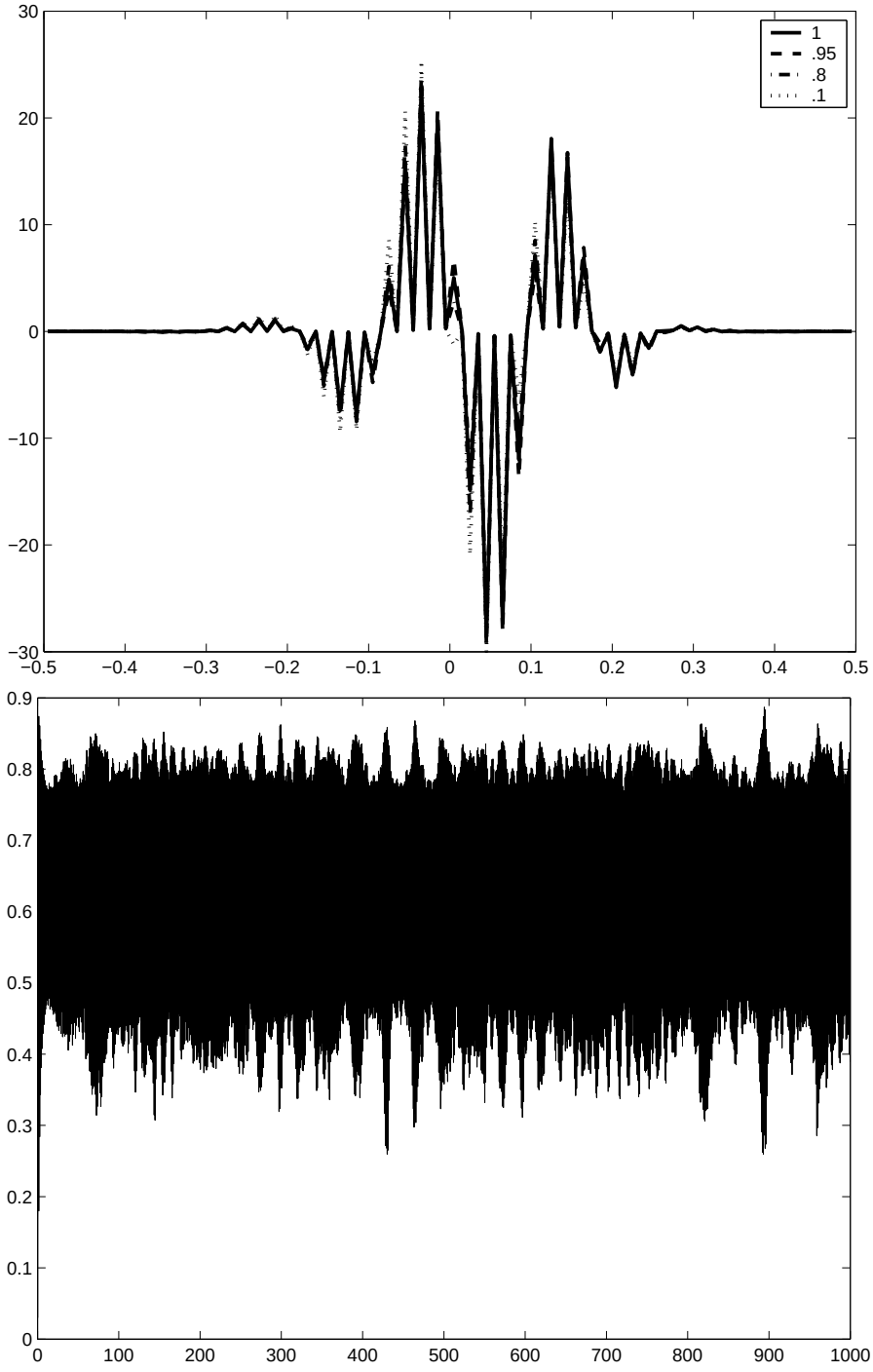

Figure 41: Crank-Nicholson method for Two-Variable Wave Equation with Dirichlet B.C. (a) One crossing time and various α; (b) Norm Plot for $\alpha = .8$.

by

$$u_t = \frac{1}{2\tau}\left((u_i^{n+1} - u_i^n) + (u_{i+1}^{n+1} - u_{i-1}^n)\right)$$

and

$$u_x = \frac{1}{2h}\left((u_{i+1}^{n+1} - u_i^{n+1}) + (u_{i+1}^n - u_i^n)\right).$$

So u_t can be approximated by

$$\frac{1}{2\tau}\left((u_i^{n+1} - u_i^n) + (u_{i+1}^{n+1} - u_{i-1}^n)\right) = \frac{-c}{2h}\left((u_{i+1}^{n+1} - u_i^{n+1}) + (u_{i+1}^n - u_i^n)\right).$$

Thus the Box method becomes

$$u_1^{n+1} = u_2^n + \left(\frac{h - c\tau}{h + c\tau}\right)(u_1^n - u_2^{n+1})$$

and similarly, using the equation $u_t = -cu_x$ at the right boundary points

$$u_N^{n+1} = u_{N-1}^n + \left(\frac{h - c\tau}{h + c\tau}\right)(u_{N-1}^{n+1} - u_N^n),$$

where N is the rightmost spatial grid point. Now we can find boundary values for u for our two systems. It turns out that v and w are both functions of $x \pm ct$ as u is for this wave equation. Thus $v_t = \pm cv_x$ and $w_t = \pm cw_x$ at the boundaries and the Box method also works for v and w.

With periodic boundary conditions, we saw that when the wave leaves the grid on the left it reappears on the right. With outgoing boundary conditions, however, once the wave leaves it does not reappear. So graphically, using the Box method for boundary points means that once the wave leaves the grid we should have $u = 0$ at all points.

5.2.1 Three-Variable System

Lax-Wendroff method Consider Figures 42-43.

The first two graphs look very similar to their analogues with Dirichlet boundary conditions, Figure 32. We see a slight difference on the righthand side of plot (a), where Dirichlet boundary conditions present a bit more jaggedness. Other than that, we see pretty much the same information – Courant factors do not matter very much, the error appears to be completely residual data from the initial wave, and the amplitude is approximately .015. Comparing the norm plots between the two boundary conditions (Figure 43 for outgoing and Figure 33 for Dirichlet), however, we notice a large difference. Dirichlet boundary conditions gave a constant magnitude of u in time, whereas outgoing boundary conditions lead our solution to grow linearly with time. Thus, for outgoing boundary conditions there is a constant nonzero value of v (the derivative of u) at all time steps in our numerical solution. This is ok though. With Dirichlet boundary conditions, the error plot will not grow as they "clamp" the value of the wave at the boundaries. Outgoing boundary conditions, on the other hand, do not require the error to be zero at any point. Having a constant value of v however is the important thing. We do not want that to change (otherwise the error would increase faster than a linear equation). With a constant v, as long as it is small, the growth only becomes significant long after the wave has left the grid.

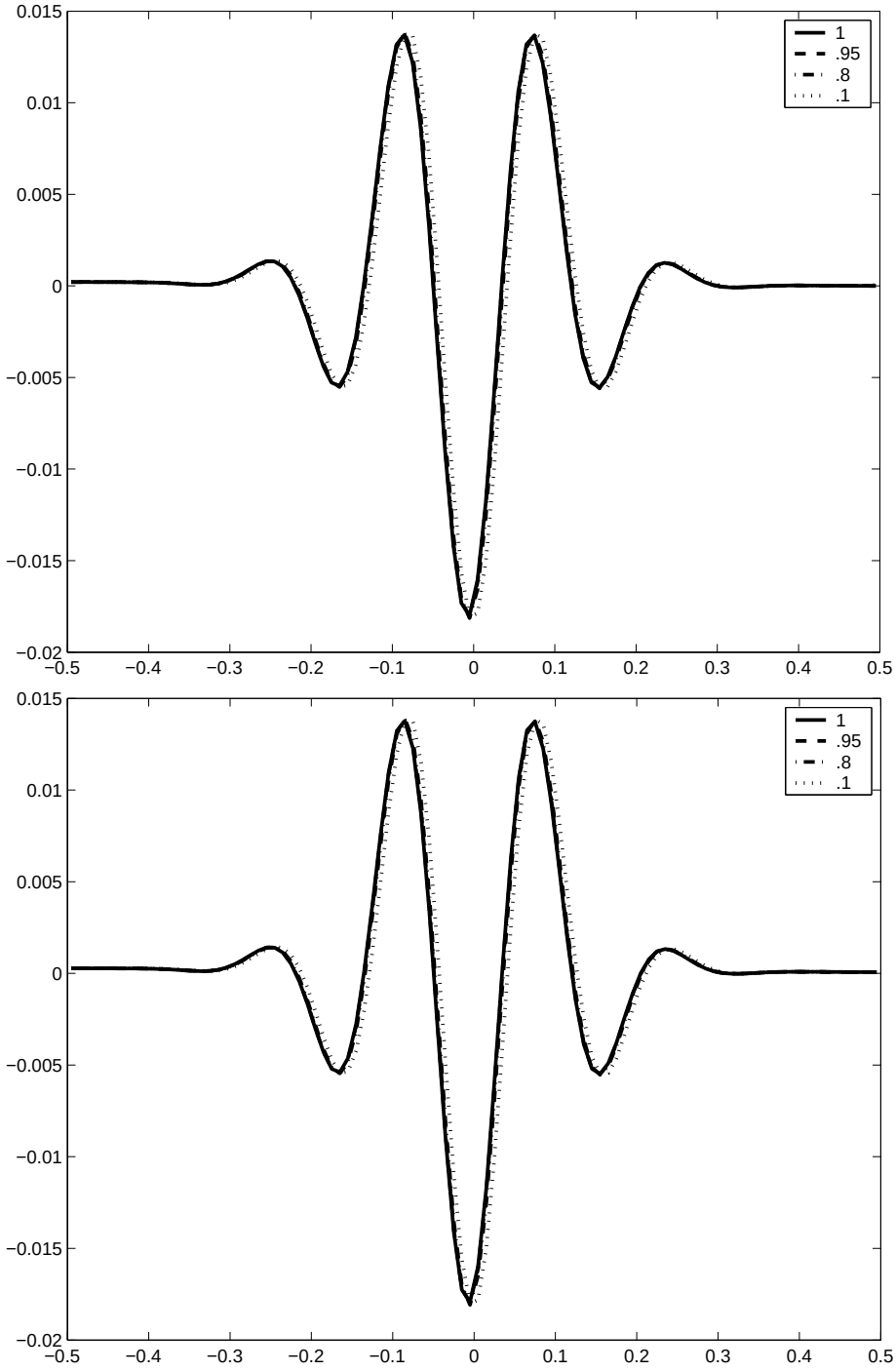

Figure 42: Lax-Wendroff method for Three-Variable Wave Equation with outgoing B.C. (a) One crossing time and various α; (b) 1.3 crossing times and various α.

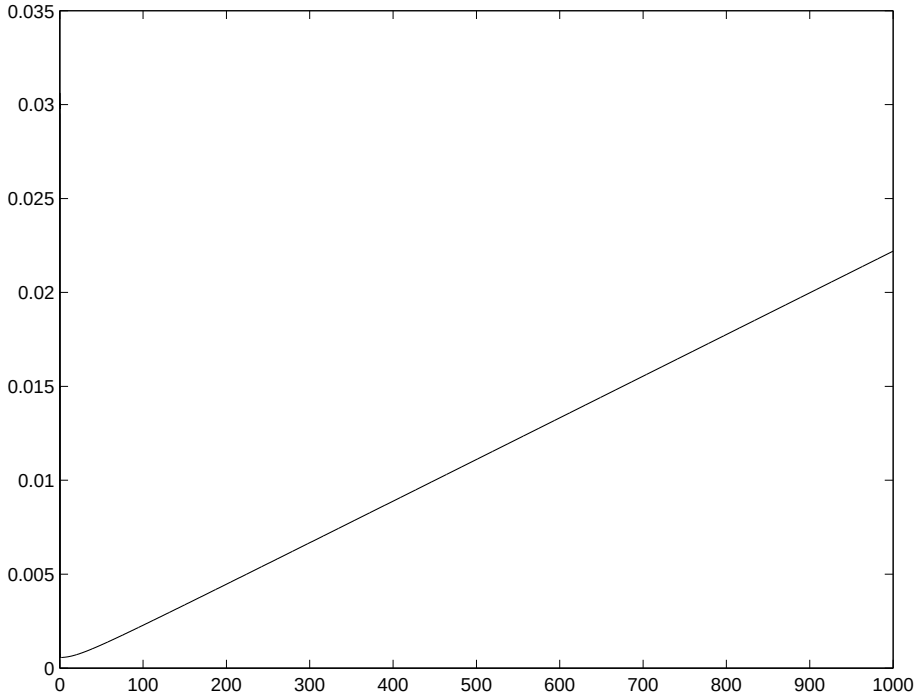

Figure 43: Norm Plot for Lax-Wendroff method for Three-Variable Wave Equation with outgoing B.C. for $\alpha = .8$.

Leapfrog method Consider Figures 44-45. Interestingly enough, outgoing boundary conditions give us a very different solution for the Leapfrog method than did Dirichlet boundary conditions. Here, the amplitude is much smaller than it was in Figure 34 by a factor of 100. Most likely as a result of the smaller magnitude, we now see a residual wave form from the initial data.

In looking at the norm plot, Figure 45, we see again that the error over time is of linear growth, which is exactly what we expect.

Iterated Crank-Nicholson method As we saw with the advection equation, the Crank-Nicholson method actually added a dispersion factor. It caused different wavelengths in the wave to travel at different speeds. It turns out that the same thing happens with the Wave Equation, but being that the solution moves in two directions, quite a bit more dispersion happens. Consider Figures 46-47.

As with the previous methods, part of the remaining wave is stationary (thus it is residual from the initial data) and the other part is actually moving in time. The latter part is again caused by a reflection from the righthand boundary that we do not want to see. Notice that the residual wave is smooth, unlike the moving wave which exhibits jaggedness, a result from reflections. Looking at the norm plot, we again see linear growth, as with the previous two methods.

5.2.2 Two-Variable System

Lax-Wendroff method Consider Figures 48-49.

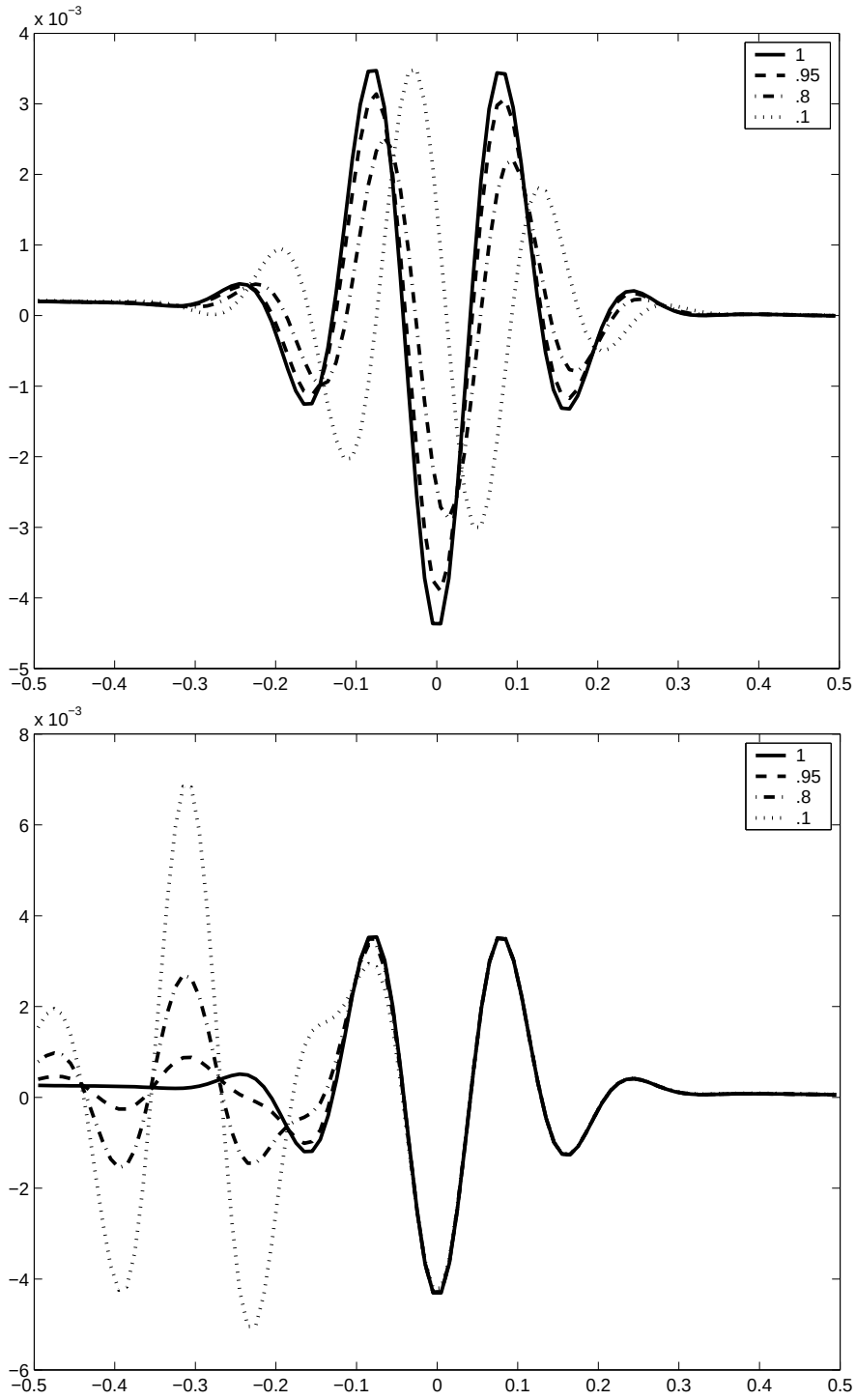

Figure 44: Leapfrog method for Three-Variable Wave Equation with outgoing B.C. (a) One crossing time and various α; (b) 1.3 crossing times and various α.

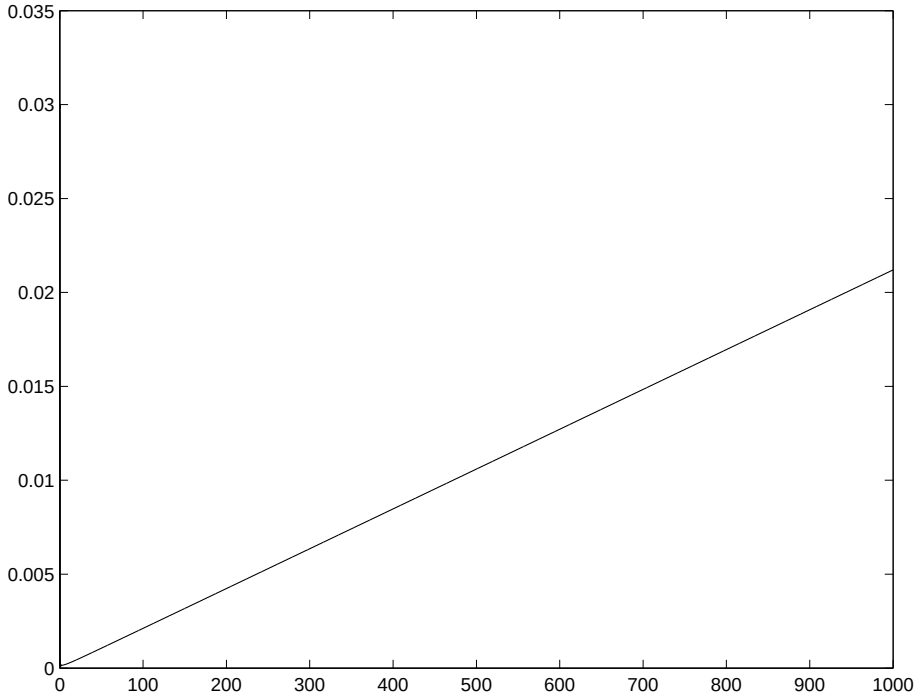

Figure 45: Norm plot for Leapfrog method for Three-Variable Wave Equation with outgoing B.C. for $\alpha = .8$.

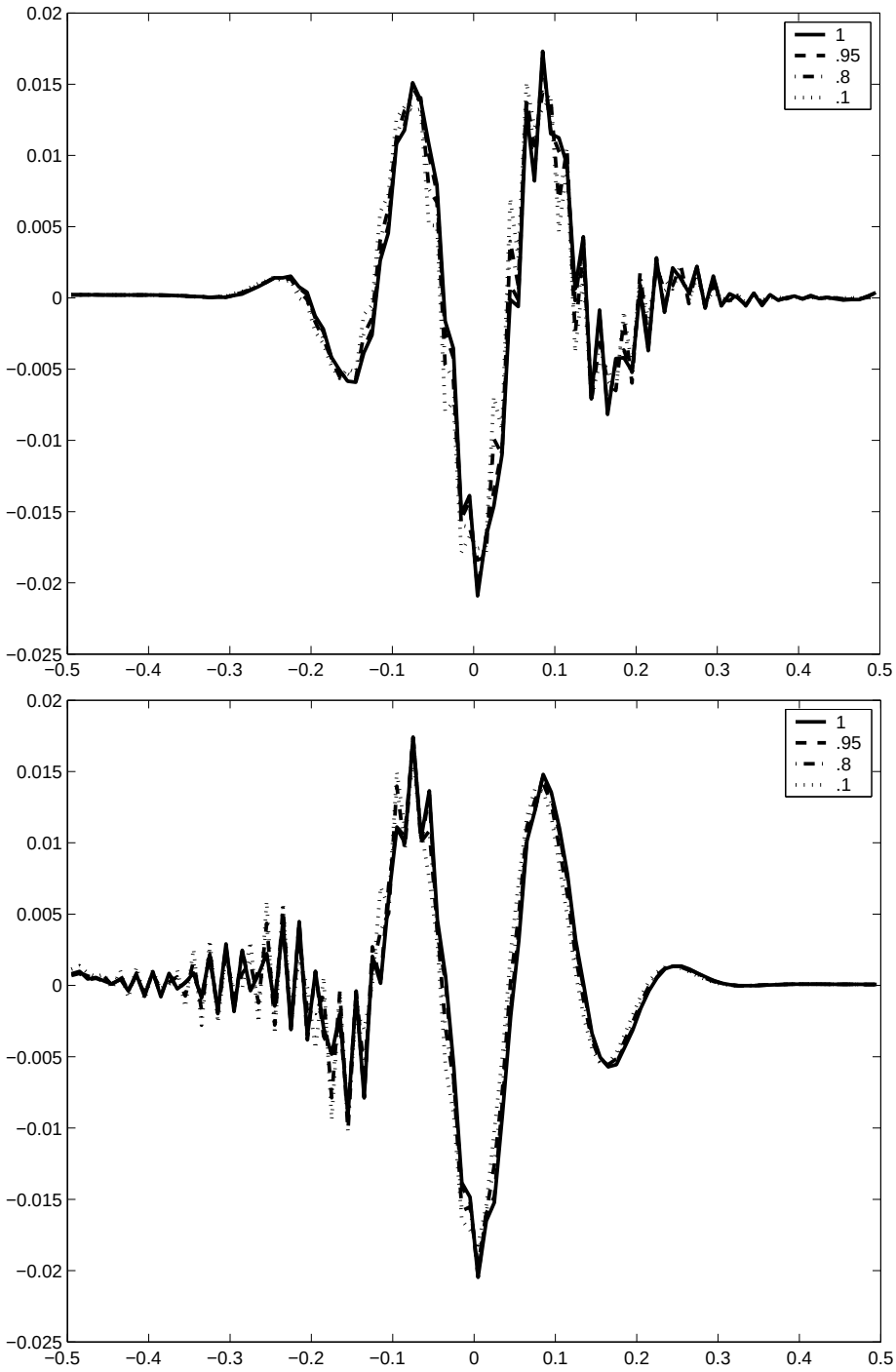

Figure 46: Crank-Nicholson method for Three-Variable Wave Equation with outgoing B.C. (a) One crossing time and various α; (b) 1.3 crossing times and various α.

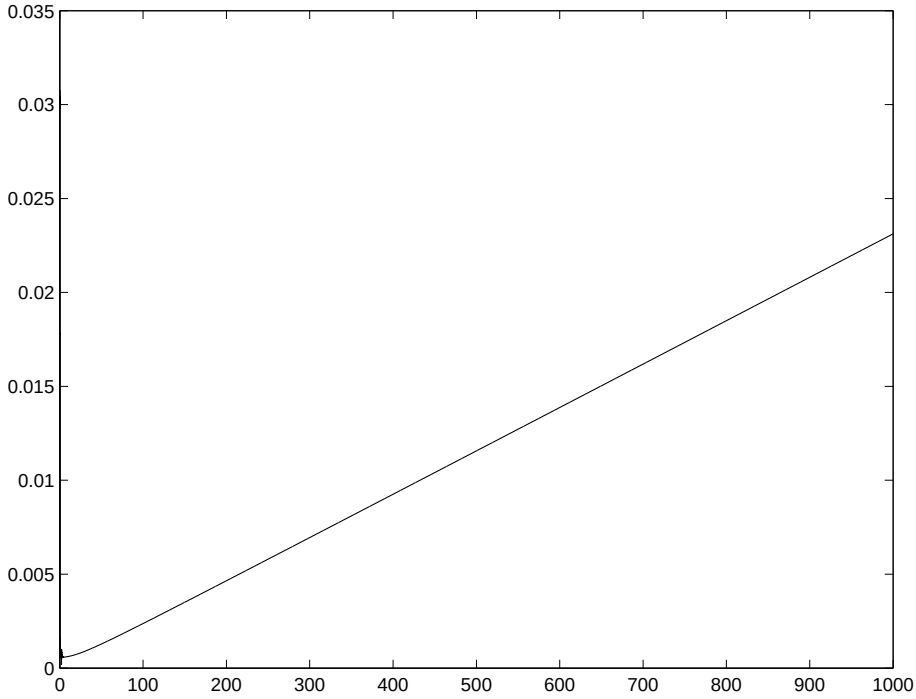

Figure 47: Norm plot for Crank-Nicholson method for Three-Variable Wave Equation with outgoing B.C. for $\alpha = .8$.

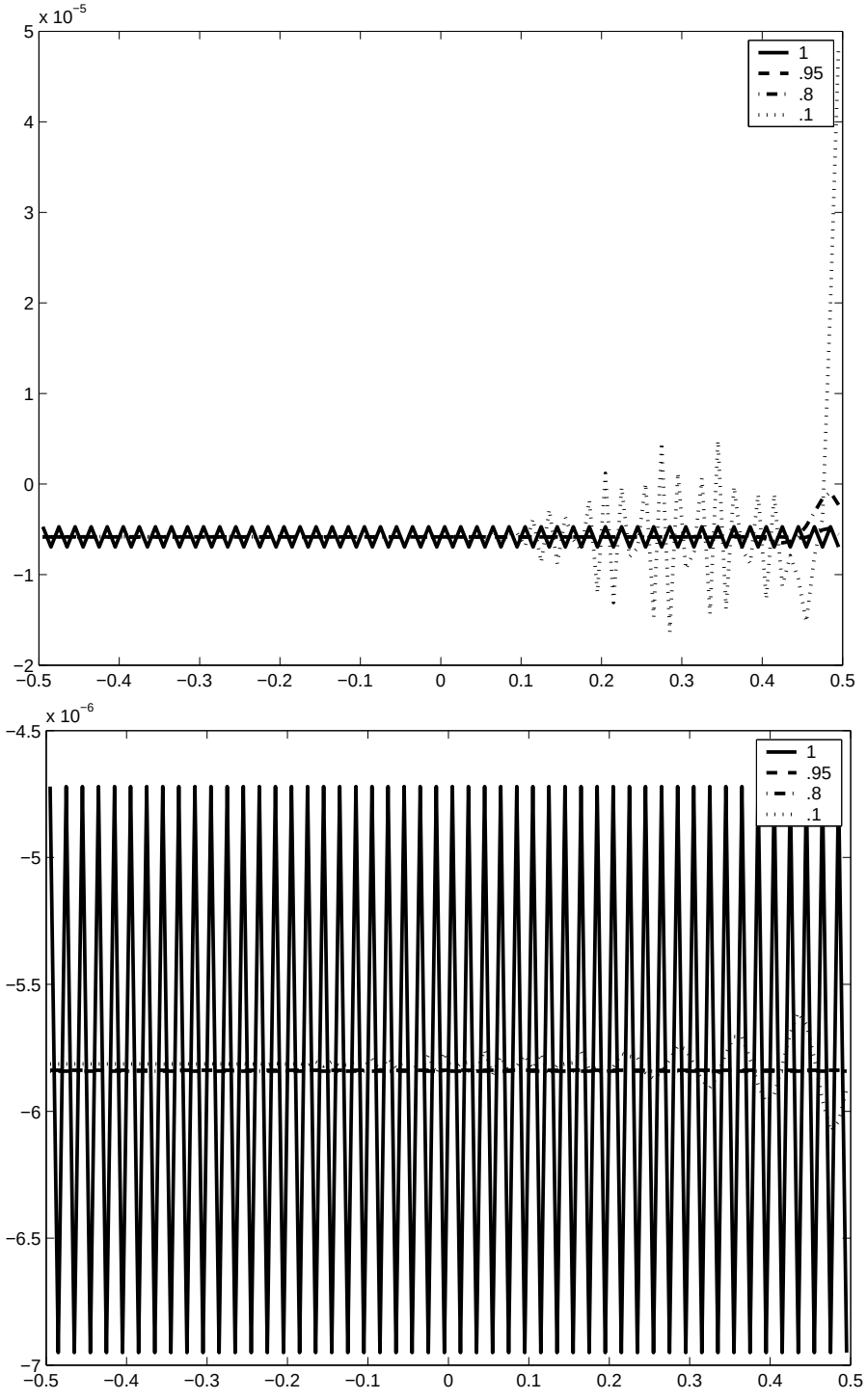

Figure 48: Lax-Wendroff method for Two-Variable Wave Equation with outgoing B.C. (a) One crossing time and various α; (b) 1.3 crossing times and various α.

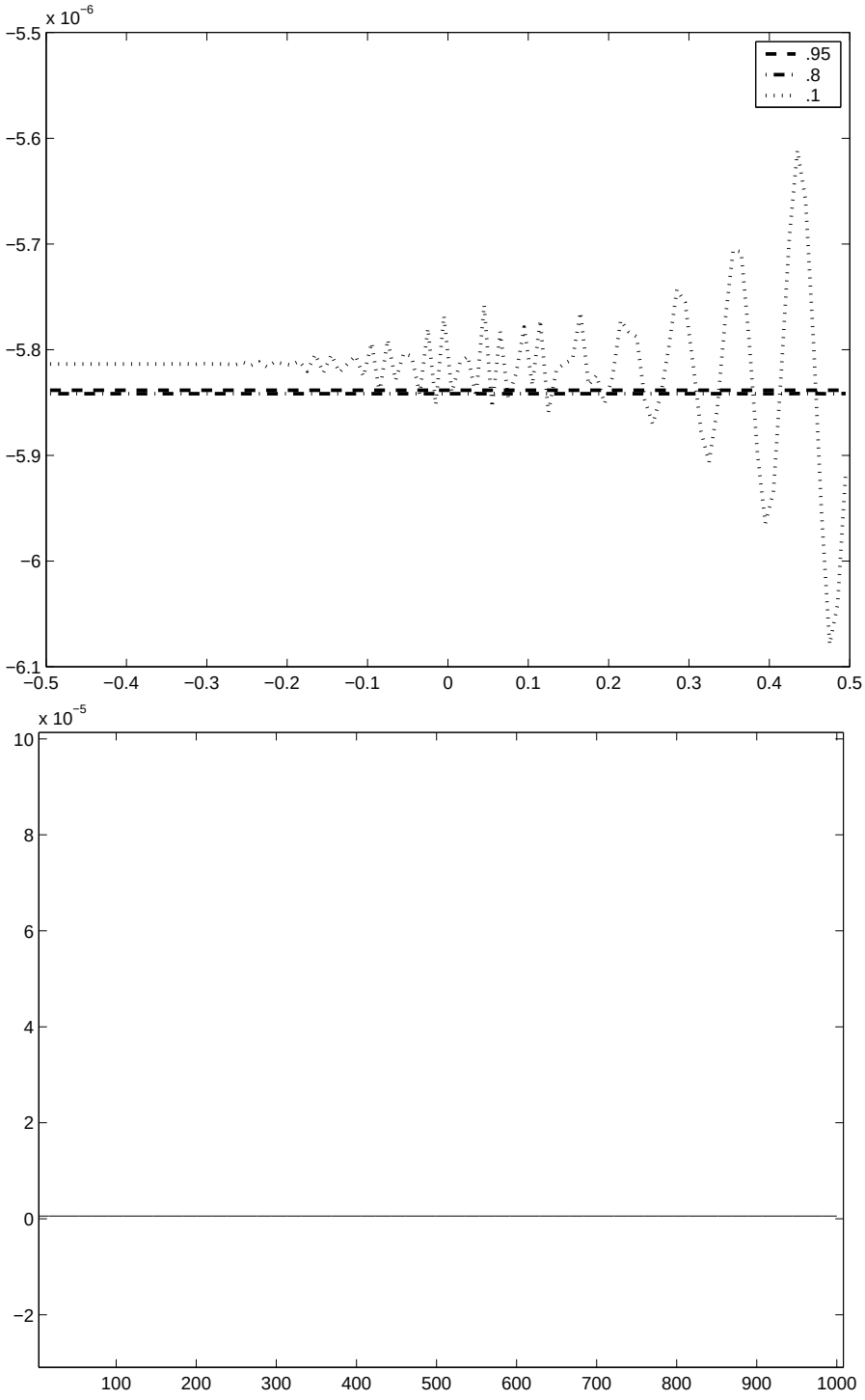

Figure 49: Lax-Wendroff method for Two-Variable Wave Equation with outgoing B.C. (c) 1.3 crossing times and various α; (d) Norm Plot for $\alpha = .8$.

Looking at plot (a), we can definitely see the error after the wave has left, and also that the error is quite small, on the order of 10^{-5}. We see that this error depends on the Courant factor so it is not a residual wave. Thus, we will look at plot (b). Plot (b) looks at the wave at 1.3 crossing times. Anything left in the same position in plots (a) and (b) would be considered a residual wave from the initial value as it would not have moved during the .3 crossing times between the two plots. Plot (b) actually shows us a smaller error (on the order of 10^{-6}). This is because the larger error in plot (a) was at the boundary and therefore left the grid due to our outgoing boundary conditions. We cannot really see if there is any residual wave so we must consider plot (c) in Figure 49, which is again the wave at 1.3 crossing times but without $\alpha = 1$. We can see that the graphs are quite different. In fact, it is important to notice that the error is larger closer to the boundary. This is because some of the wave is not actually leaving the grid, but is instead being reflected backwards. Thus the Lax-Wendroff method has small reflections at the boundary.

Consider the magnitude of these plots. Again, they are smaller than with Dirichlet boundary conditions. In fact, they are smaller by a factor of 10^7. We are seeing a trend between outgoing and Dirichlet boundary conditions: as far as minimizing the magnitude of the wave, outgoing boundary conditions are better.

We now want to consider the value of u over time. Looking at Figure 49(d), we see that the norm is essentially zero. This is great! We want our solution to be zero after it has left the plot. We did not expect this with periodic boundary conditions because the solution never actually left the grid– it circled around (as

if on a cylinder) instead. Thus Lax-Wendroff is stable in the two-variable system and in fact has a constant error as opposed to the linear growth we saw with the three-variable system.

Leapfrog method Consider Figure 50.

Looking at plot (a), we see what appears to be the inverse of the wave for Dirichlet boundary conditions(Figure 40), except that the amplitude is again smaller by a factor of 100. This is not surprising. Every method we have seen thus far (except for the unstable Crank-Nicholson method) has a smaller amplitude with outgoing boundary conditions than with Dirichlet boundary conditions. Looking at the norm plot in Figure 50, we see again that the norm is essentially zero (or at least on the order of 10^{-6}).

Iterated Crank-Nicholson method Looking at the numerical solution in Figure 51(a), we see that the Crank-Nicholson method gives us more jaggedness than the previous two methods. Of course, as Figure 51(b) shows, the magnitude of u is still very nearly zero. Comparing outgoing with Dirichlet boundary conditions, we see that Crank-Nicholson gives a slightly better approximation with outgoing conditions.

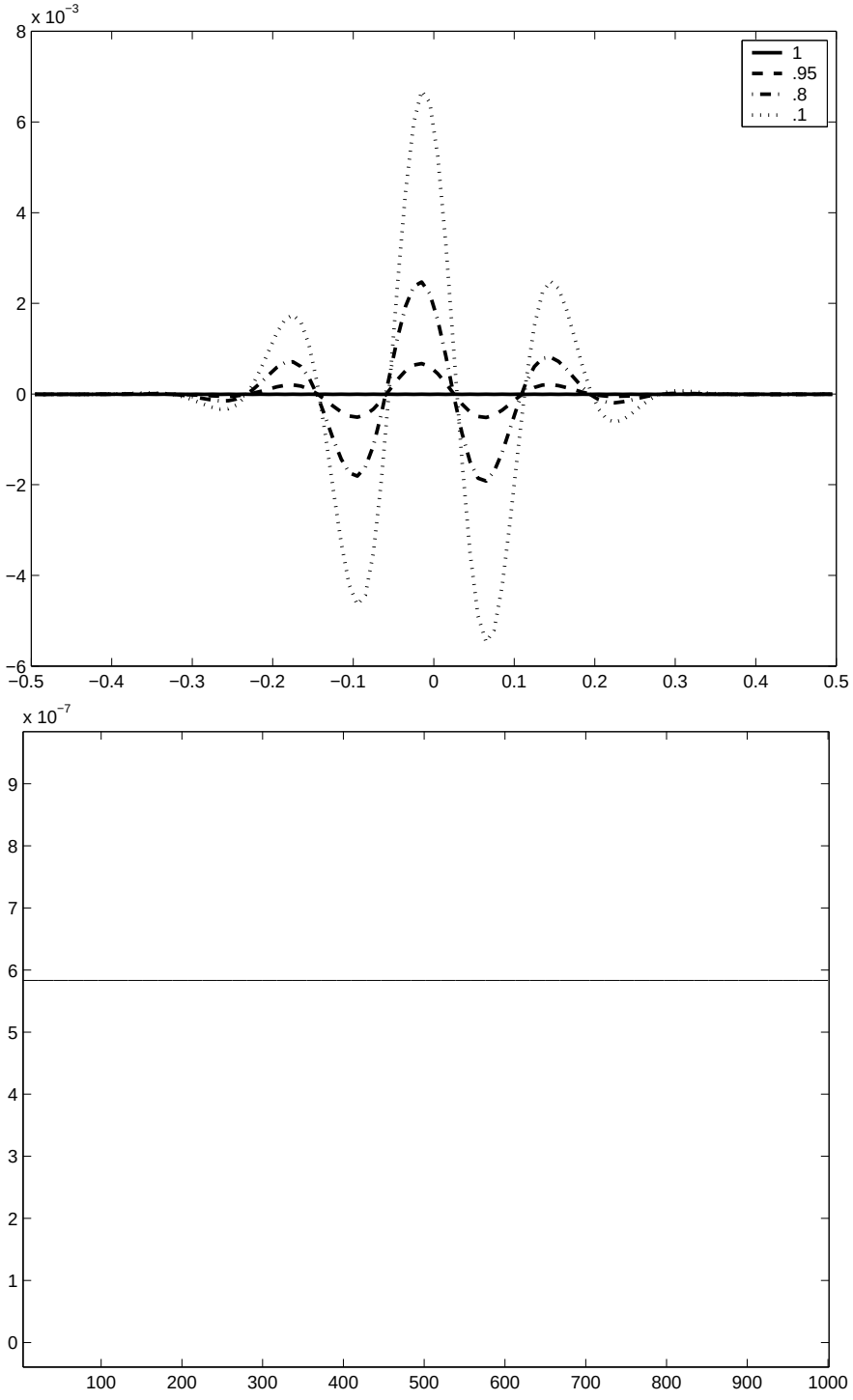

Figure 50: Leapfrog method for Two-Variable Wave Equation with outgoing B.C. (a) One crossing time and various α; (b)Norm plot for $\alpha = .8$.

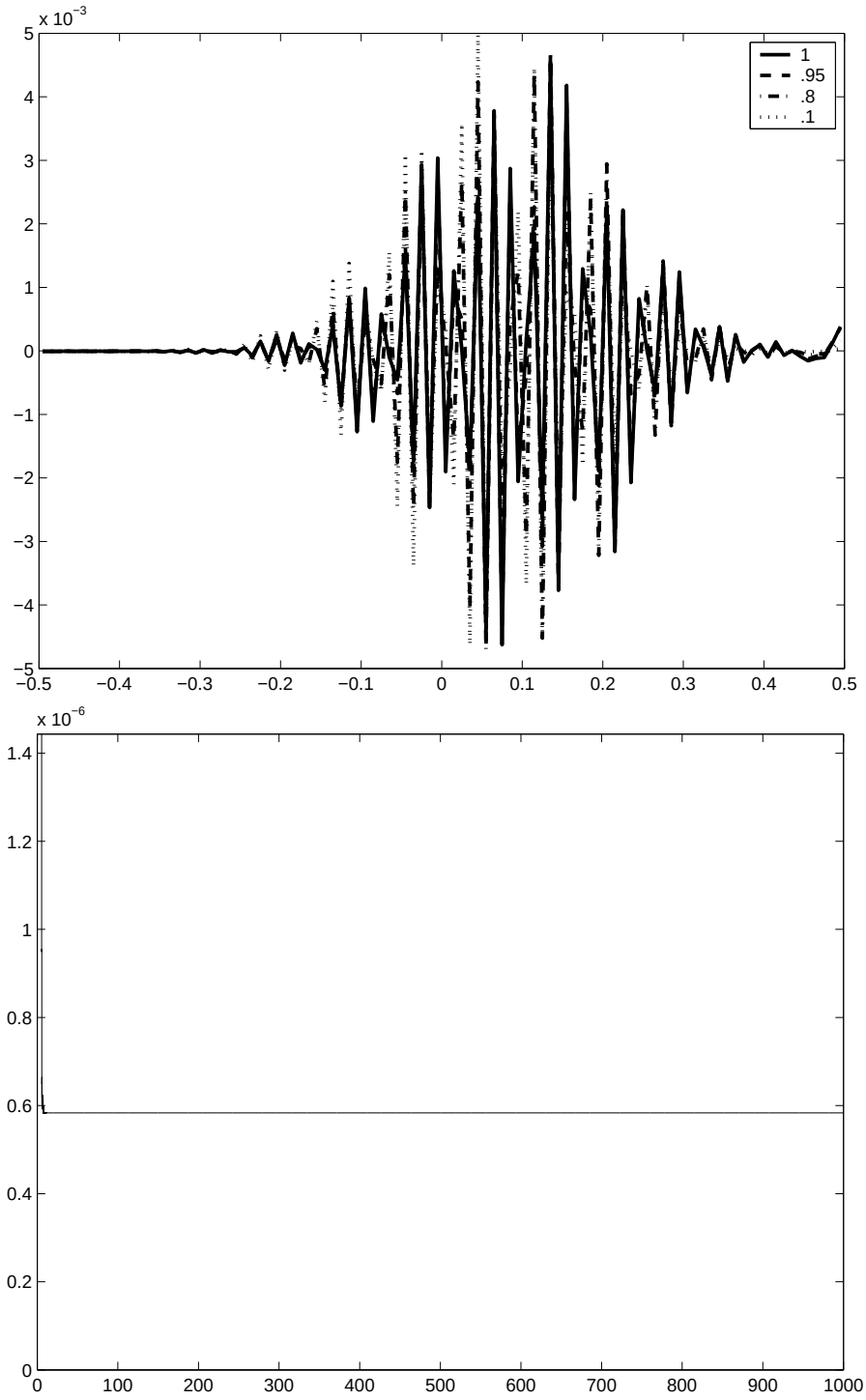

Figure 51: Crank-Nicholson method for Two-Variable Wave Equation with outgoing B.C. (a) One crossing time and various α; (b) Norm Plot for $\alpha = .8$.

5.3 Summary

We have analyzed both the different methods and the various boundary conditions. We have seen strengths and weaknesses in both Lax-Wendroff and Leapfrog. Both methods give better approximations than Crank-Nicholson. This is no surprise as we have seen this all along.

Comparing the various norm plots between the methods, we see that outgoing boundary conditions tend to give better results than Dirichlet boundary conditions for the two-variable system and exactly the opposite occurs for the three-variable system. The advantage of outgoing boundary conditions is that it relies on properties of the outgoing wave and thus does not require an analytical solution. However, if we have the analytical solution, Dirichlet is a good method to use. Of course, there is also periodic boundary conditions, which have the advantage of allowing us to see dissipation and dispersion over time.

6 Nonlinear Systems

In [HKN], the authors present the nonlinear hyperbolic partial differential equation

$$g_{tt} = g_{xx} - \frac{g_t^2}{g}.$$

This equation has nonlinearity similar to that of general relativity, and, in particular, Einstein's equations. By studying this system, we can learn about the strengths and weaknesses of each method as applied to general relativity. The authors of [HKN] show the stability for four different numerical methods, the Iterative Crank-Nicholson, third-order Runge-Kutta, fourth-order Runge-Kutta, and the Courant-Friedrichs-Levy Nonlinear (CFLN) schemes. The CFLN method is similar to Leapfrog, however there is a difference in their evaluation of g_t. In our study, we will look at the three methods we have studied over the past three chapters, with Dirichlet boundary conditions.

We will use the exponential growth function

$$g(x,t) = e^{x+t/\sqrt{2}}$$

as our initial condition. This is a solution of the nonlinear wave equation as

$$g_{tt} = .5g$$

and

$$g_{xx} - \frac{g_t^2}{g} = g - \frac{.5g^2}{g} = .5g.$$

Unfortunately, by using the exponentially growing solution, we will not be able to consider the types of graphs we have considered previously. The wave, and therefore

the norm, grow exponentially, as we expect the error to. Thus, we we will look at plots of the relative error, that is, plots of

$$\frac{\text{calculated value} - \text{actual value}}{\text{actual value}}$$

where the calculated value is the value of the function given by our numerical method and the actual value is

$$g(x,t) = e^{x+t/\sqrt{2}}.$$

As with the wave equation, we will again look at two different systems that are equivalent to our wave. The first system is

$$\left|\begin{array}{l} g_t = K \\ K_t = w_x - \frac{g_t^2}{g} \\ w_t = K_x \end{array}\right.$$

and the second system is

$$\left|\begin{array}{l} g_t = K \\ K_t = g_{xx} - \frac{g_t^2}{g} \end{array}\right.$$

6.1 Three-Variable System

6.1.1 Lax-Wendroff Method

How It Works Recall that Lax-Wendroff approximates g by the second-order Taylor series, where the derivatives are replaced by there difference equation equivalents. Computing the first- and second-order time derivatives of each variable, we

get the difference equations. Since

$$g_t = K,$$

$$g_{tt} = w_x - \frac{K^2}{g},$$

$$K_t = w_x - \frac{K^2}{g},$$

$$K_{tt} = K_{xx} - \frac{2KK_t}{g} + \frac{K^3}{g^2},$$

$$w_t = K_x, and$$

$$w_{tt} = K_{xt} = K_{tx} = w_{xx} - \frac{2KK_x}{g} + \frac{Kw}{g^2},$$

we get the difference equations

$$g_i^{n+1} = g_i^n + \tau g_t + \frac{\tau}{2} g_{tt},$$

$$K_i^{n+1} = K_i^n + \tau K_t + \frac{\tau}{2} K_{tt}, \text{ and}$$

$$w_i^{n+1} = w_i^n + \tau w_t + \frac{\tau}{2} w_{tt},$$

where

$$\begin{aligned}
g_t &= K_i^n, \\
g_{tt} &= \frac{w_{i+1}^n - w_{i-1}^n}{2h} - \frac{(K_i^n)^2}{g_i^n}, \\
K_t &= \frac{w_{i+1}^n - w_{i-1}^n}{2h} - \frac{(K_i^n)^2}{g_i^n}, \\
K_{tt} &= \frac{K_{i+1}^n - K_{i-1}^n}{2h} - \left(\frac{2K_i^n}{g_i^n}\right)\left(\frac{w_{i+1}^n - w_{i-1}^n}{2h} - \frac{(K_i^n)^2}{g_i^n}\right) + \frac{(K_i^n)^3}{(g_i^n)^2}, \\
w_t &= \frac{K_{i+1}^n - K_{i-1}^n}{2h}, \text{ and} \\
w_{tt} &= \frac{w_{i+1}^n - 2w_i^n + w_{i-1}^n}{h^2} - \left(\frac{2K_i^n}{g_i^n}\right)\left(\frac{K_{i+1}^n - K_{i-1}^n}{2h}\right) + \frac{K_i^n w_i^n}{(g_i^n)^2}.
\end{aligned}$$

Solutions Consider Figure 52.

We see for a Courant factor of .8, our method is stable up to around 330 crossing times. At this point, the graph seems to disappear, but not because of instability.

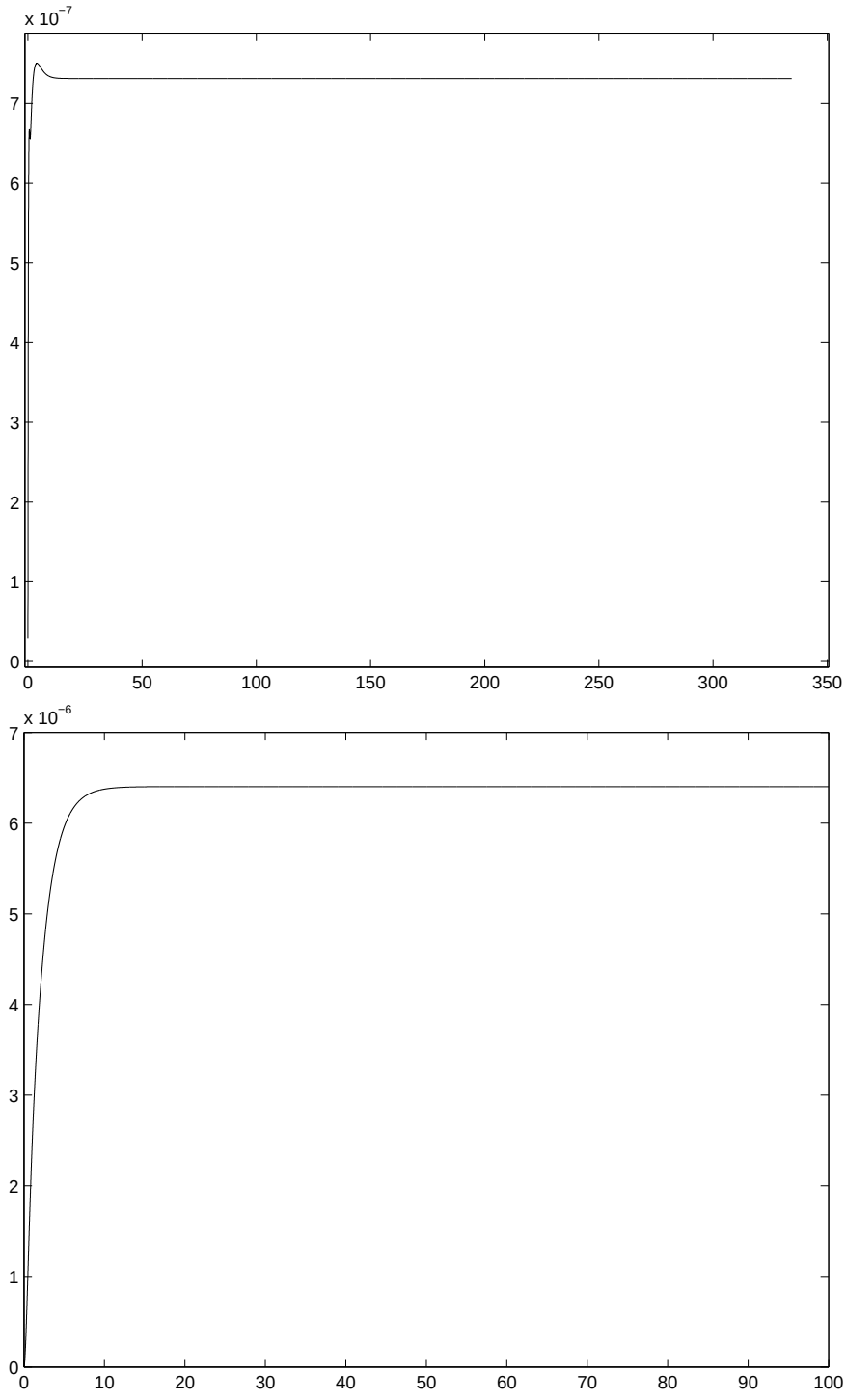

Figure 52: Lax-Wendroff method for Three-Variable Nonlinear Equation (a) $\alpha = .8$; (b) $\alpha = .1$.

This is because the values of the function are becoming too large for the computer to handle. Also notice the magnitude of the error: 10^{-7}. There is very little difference between the two functions! Thus, for all purposes, Lax-Wendroff is stable for our nonlinear system (with a Courant factor of .8). Let's look at how Leapfrog models our solution.

6.1.2 Leapfrog Method

How It Works Recall from the wave equation that we used half-steps for the Leapfrog method. We will again use half-steps for our nonlinear system. In fact, we will use the same grid system as in the three-variable wave equation. That is, our lattice will be constructed so that we find the values of g at the lattice points (i, n), the values of K at the lattice points $(i, n + \frac{1}{2})$, and the values of w at the lattice points $(i + \frac{1}{2}, n)$. Thus our difference equations will be

$$g_i^{n+1} = g_i^{n-1} + 2\tau \left(K_i^{n+.5}\right),$$

$$K_i^{n+.5} = K_i^{n-.5} + 2\tau \left(\frac{w_{i+.5}^n - w_{i-.5}^n}{h} - \frac{(K_i^{n+.5})^2}{g_i^n}\right), \text{ and}$$

$$w_{i+.5}^{n+1} = w_{i+.5}^{n-1} + 2\tau \left(\frac{K_{i+.5}^n - K_{i-.5}^n}{h}\right).$$

Solutions Unfortunately, Leapfrog is not as stable for as long as Lax-Wendroff. Consider Figure 53.

For a Courant factor of .8, our solution blows up after about 50 crossing times. In plot (b) we see that Leapfrog is stable for much longer. Up to 100 crossing

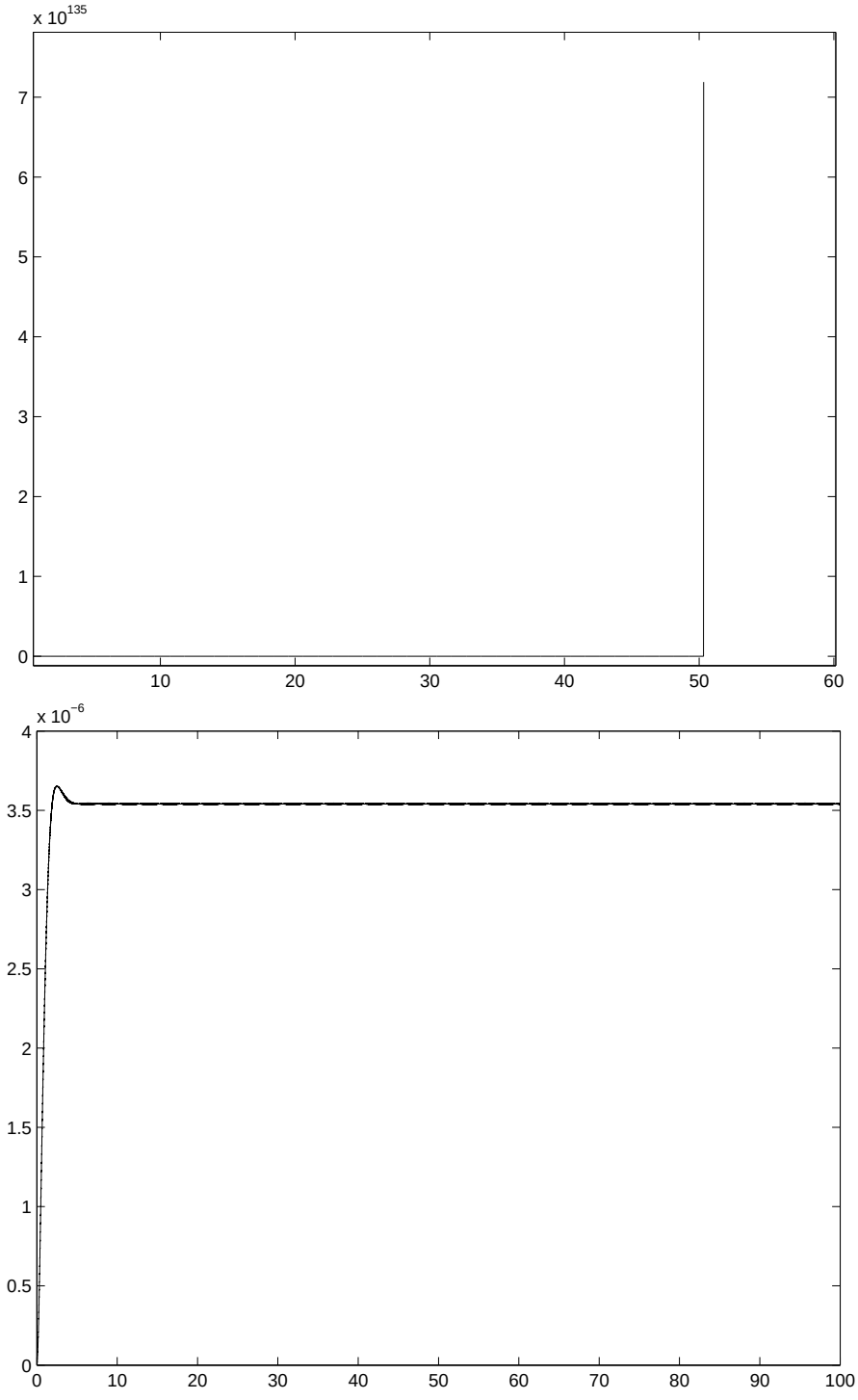

Figure 53: Leapfrog method for Three-Variable Nonlinear Equation (a) $\alpha = .8$; (b) $\alpha = .1$.

times, Leapfrog models our exponential function quite accurately with an error on the order of 10^{-6} (not as small as Lax-Wendroff, but still significantly small). We could check beyond 100 crossing times; however, using such a small Courant factor means a much longer time to produce results. In fact, it takes the same amount of time to run a plot for $\alpha = .1$ for 100 crossing times as it takes to run a plot for $\alpha = .8$ for 1000 crossing times. Let's see how Crank-Nicholson behaves, using our nonlinear equation.

6.1.3 Iterative Crank-Nicholson Method

How It Works Recall from before that for the Iterated Crank-Nicholson method, we took each term in the second term on the righthand side of the FTCS method and replaced it by the average of that value and the value of the function at the next time step. Thus our difference equations are

$$g_i^{n+1} = g_i^n + \frac{\tau}{2}\left(K_i^{n+1} + K_i^n\right),$$
$$K_i^{n+1} = K_i^n + \frac{\tau}{2}\left(\frac{w_{i+1}^{n+1} - w_{i-1}^{n+1}}{2h} - \frac{(K_i^{n+1})^2}{g_i^{n+1}} + \frac{w_{i+1}^n - w_{i-1}^n}{2h} - \frac{(K_i^n)^2}{g_i^n}\right),$$
$$w_i^{n+1} = w_i^n + \frac{\tau}{2}\left(\frac{K_{i+1}^{n+1} - K_{i-1}^{n+1}}{2} + \frac{K_{i+1}^n - K_{i-1}^n}{2}\right),$$

Solutions For a Courant factor of .8, Figure 54(a) shows that Crank-Nicholson blows up at only 4.5 crossing times! This is a very short amount of time, especially compared to our previous two methods. In plot (b), however, we see that Crank-Nicholson gives the same amount of error as Leapfrog. Perhaps the stability condition is stricter than $\alpha \leq 1$ for the Leapfrog and Crank-Nicholson methods.

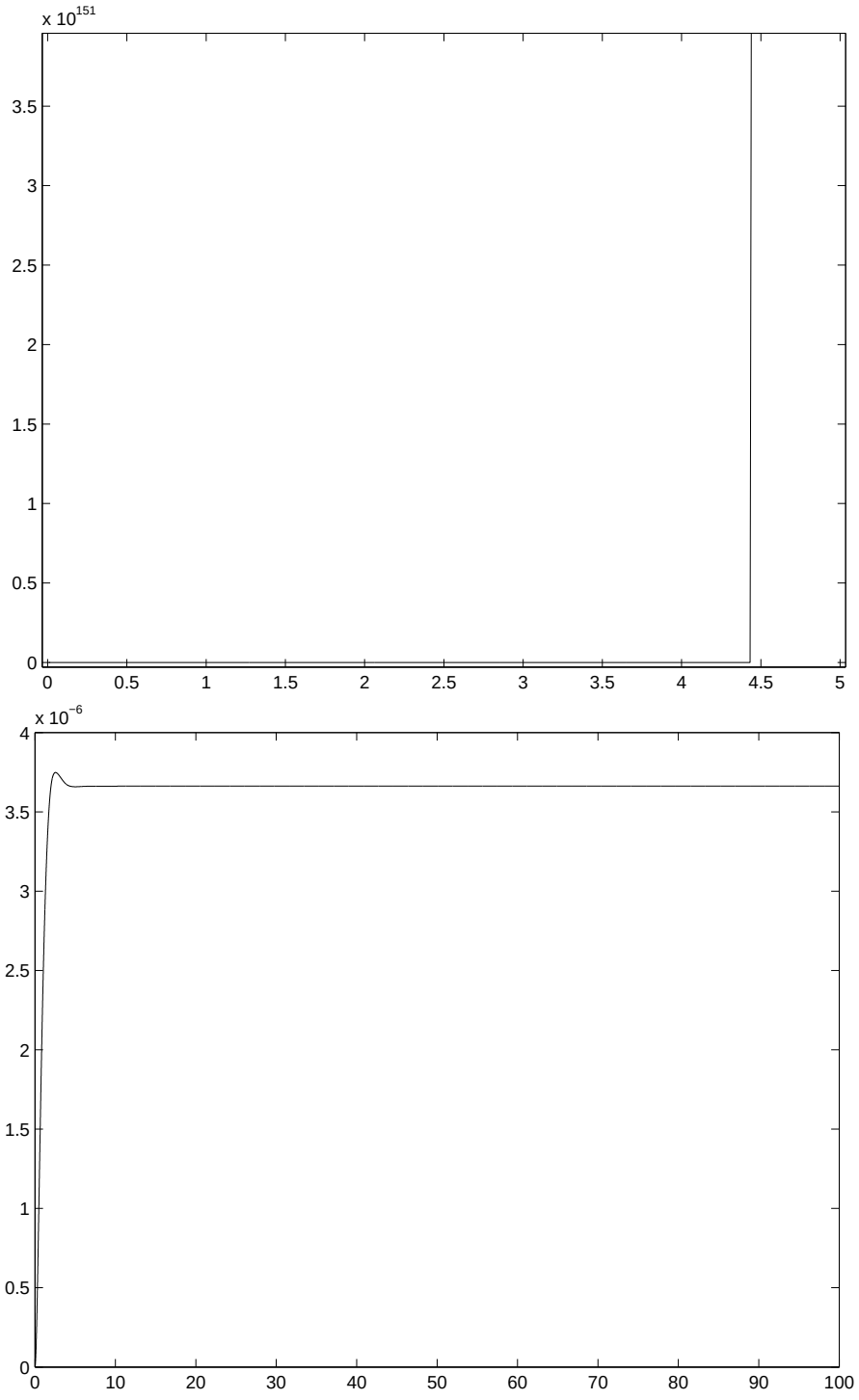

Figure 54: Crank-Nicholson method for Three-Variable Nonlinear Equation (a) $\alpha = .8$; (b) $\alpha = .1$.

Let's see how the two-variable system shapes up.

6.2 Two-Variable System

6.2.1 Lax-Wendroff Method

How It Works Recall that Lax-Wendroff approximates g by the second-order Taylor series, where the derivatives are replaced by there difference equation equivalents. Computing the first- and second-order time derivatives of each variable, we get the difference equations

$$g_i^{n+1} = g_i^n + \tau g_t + \frac{\tau}{2} g_{tt}, \text{ and}$$

$$K_i^{n+1} = K_i^n + \tau K_t + \frac{\tau}{2} K_{tt},$$

where

$$\begin{aligned}
g_t &= K_i^n, \\
g_{tt} &= \tfrac{g_{i+1}^n - 2g_i^n - g_{i-1}^n}{h^2} - \tfrac{(K_i^n)^2}{g_i^n}, \\
K_t &= \tfrac{g_{i+1}^n - 2g_i^n - g_{i-1}^n}{h^2} - \tfrac{(K_i^n)^2}{g_i^n}, \\
K_{tt} &= \tfrac{K_{i+1}^n - K_{i-1}^n}{2h} - \left(\tfrac{2K_i^n}{g_i^n}\right)\left(\tfrac{g_{i+1}^n - 2g_i^n - g_{i-1}^n}{h^2} - \tfrac{(K_i^n)^2}{g_i^n}\right) + \tfrac{(K_i^n)^3}{(g_i^n)^2}.
\end{aligned}$$

Solutions Compare Figure 55 below to Figure 52 above.

Interestingly, although the three-variable system is stable for higher Courant factors, the relative error for $\alpha = .1$ is smaller for the two-variable system. In Figure 55(a), we see that the solution blows up before .5 crossing times whereas in Figure 52, the solution was steady up to 330 crossing times. Of course, as with the

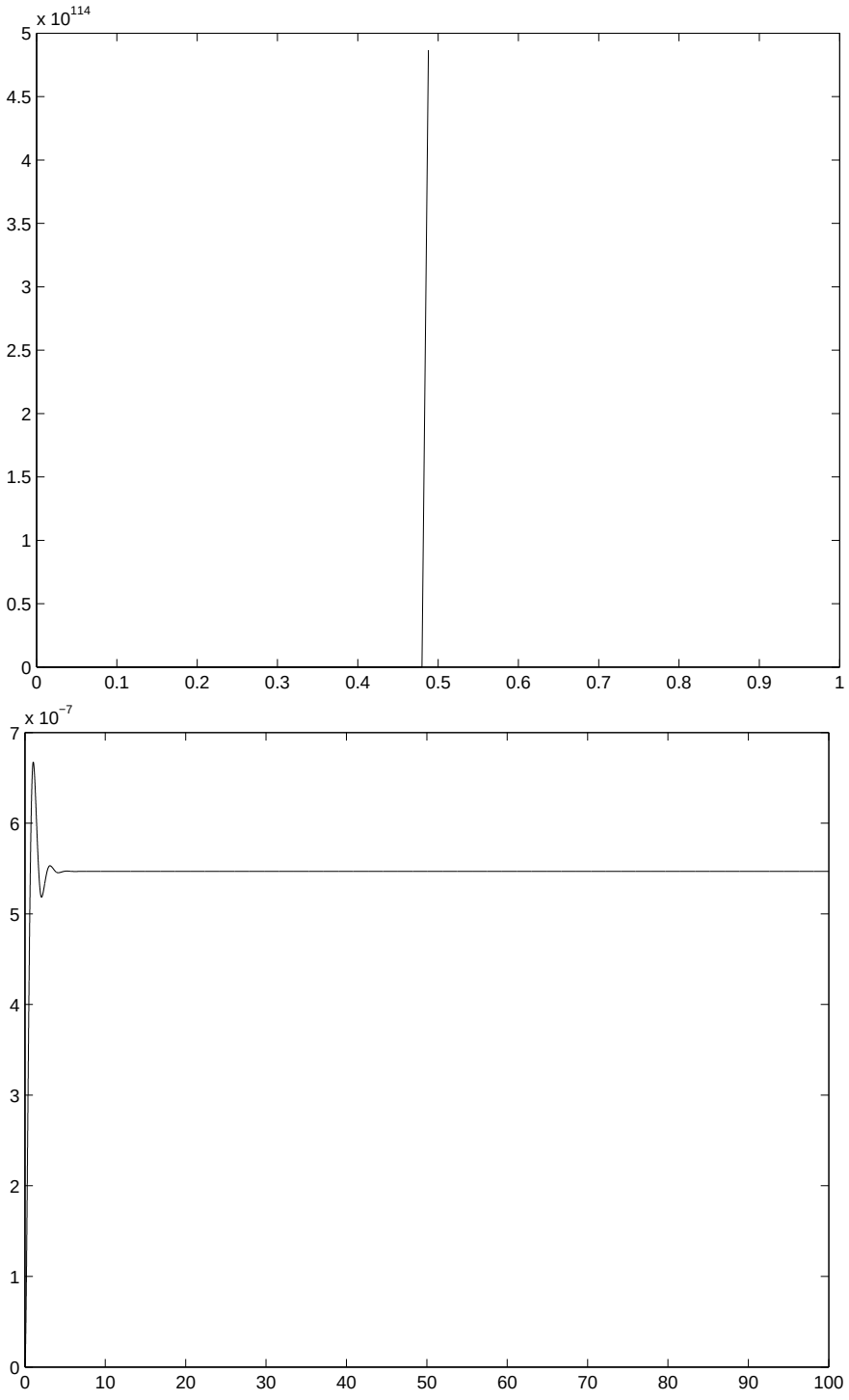

Figure 55: Lax-Wendroff method for Two-Variable Nonlinear Equation (a) $\alpha = .8$; (b) $\alpha = .1$.

previous methods, the solution is stable and has a very small magnitude of error for a Courant factor of .1.

6.2.2 Leapfrog Method

How It Works Unfortunately, we have a second derivative in this system. Thus we must evaluate both g and K on whole steps, instead of using half steps as we have in previous methods. Thus our difference equations are

$$g_i^{n+1} = g_i^{n-1} + 2\tau K_i^n \text{ and}$$

$$K_i^{n+1} = K_i^{n-1} + 2\tau \left(\frac{g_{i+1}^n - 2g_i^n + g_{i-1}^n}{h^2} - \frac{(K_i^n)^2}{g_i^n} \right).$$

Solutions Again, we see that the three-variable system is stable for higher courant factors but that the two-variable system has a smaller relative error for $\alpha = .1$. In Figure 56(a), the solution blows up before .6 crossing times as opposed to the 50 crossing times it took in Figure 53(a). The important thing is, however, that Leapfrog is stable (at least to 100 crossing times) for a small Courant factor (Figure 56(b)).

6.2.3 Iterative Crank-Nicholson Method

How It Works Again recall that for the Iterated Crank-Nicholson method, we took each term in the second term on the right hand side of the FTCS method and replaced it by the average of that value and the value of the function at the next

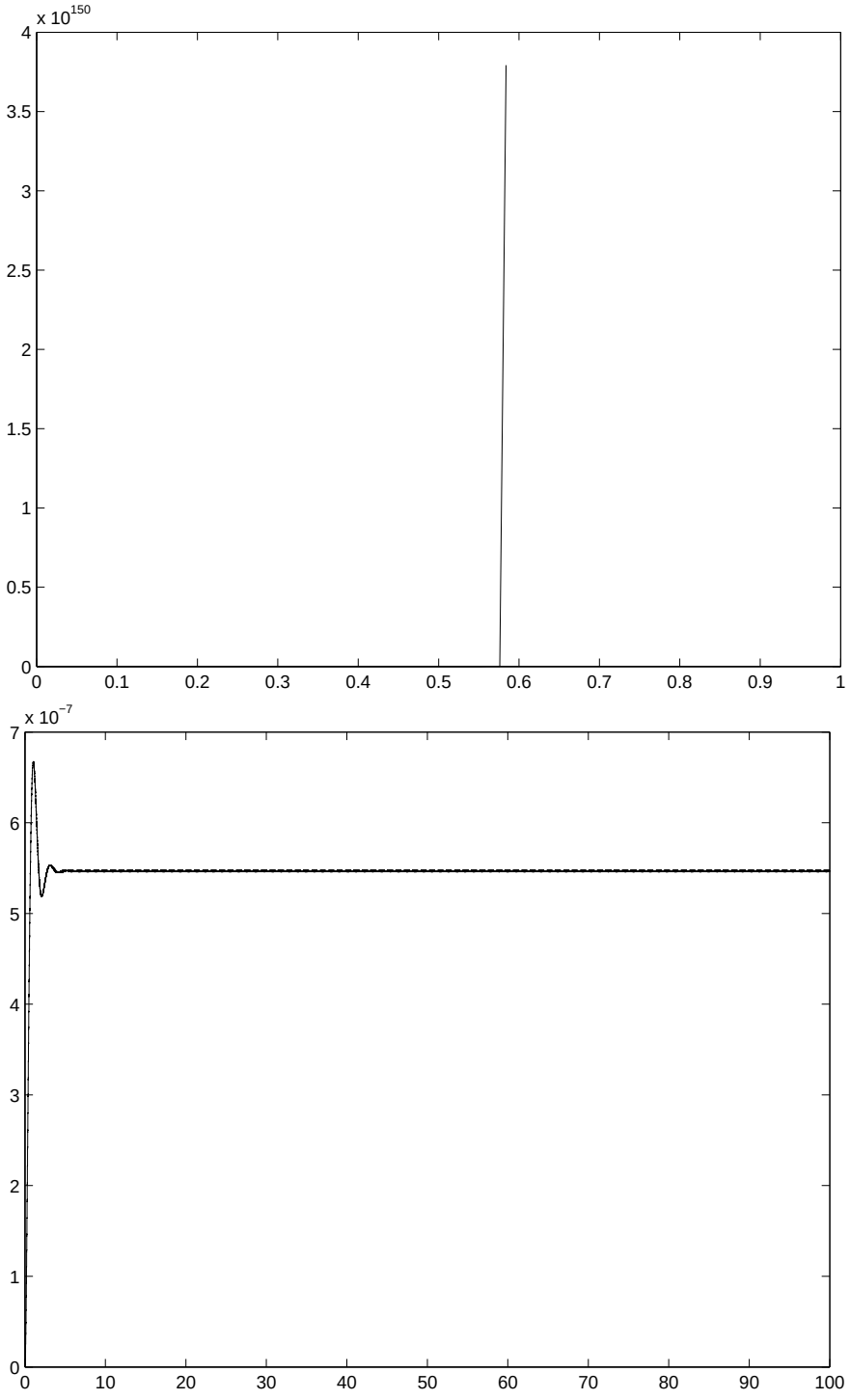

Figure 56: Leapfrog method for Two-Variable Nonlinear Equation (a) $\alpha = .8$; (b) $\alpha = .1$.

time step. Thus our difference equations are

$$g_i^{n+1} = g_i^n + \frac{\tau}{2}\left(K_i^{n+1} + K_i^n\right) \text{ and}$$

$$g_i^{n+1} = g_i^n + \frac{\tau}{2}\left(\frac{g_{i+1}^{n+1} - 2g_i^{n+1} + g_{i-1}^{n+1}}{h^2} - \frac{(K_i^{n+1})^2}{g_i^{n+1}} + \frac{g_{i+1}^n - 2g_i^n + g_{i-1}^n}{h^2} - \frac{(K_i^n)^2}{g_i^n}\right)$$

Solutions Similar to the previous results, the three-variable system is better for a Courant factor of .8. However, this time the two-variable system is worse for a Courant factor of .1. Consider Figure 57(b), though.

Notice the magnitude of error. The error is on the order of 10^{-2}, which is much larger than the 10^{-6} errors we were seeing previously. Thus, Crank-Nicholson is not such a great approach for the two-variable system.

6.3 Summary

In all of the methods, we saw that the three-variable system was a better approximation that the two-variable system for a Courant factor of .8 and therefore is more stable at higher Courant factors. However, the two-variable system was a better approximation at a Courant factor of .1 for the Lax-Wendroff and Leapfrog methods. This most likely has to do with the second-derivative in the latter system. However, for all practical purposes, all of the methods above would work, provided we used a small Courant factor.

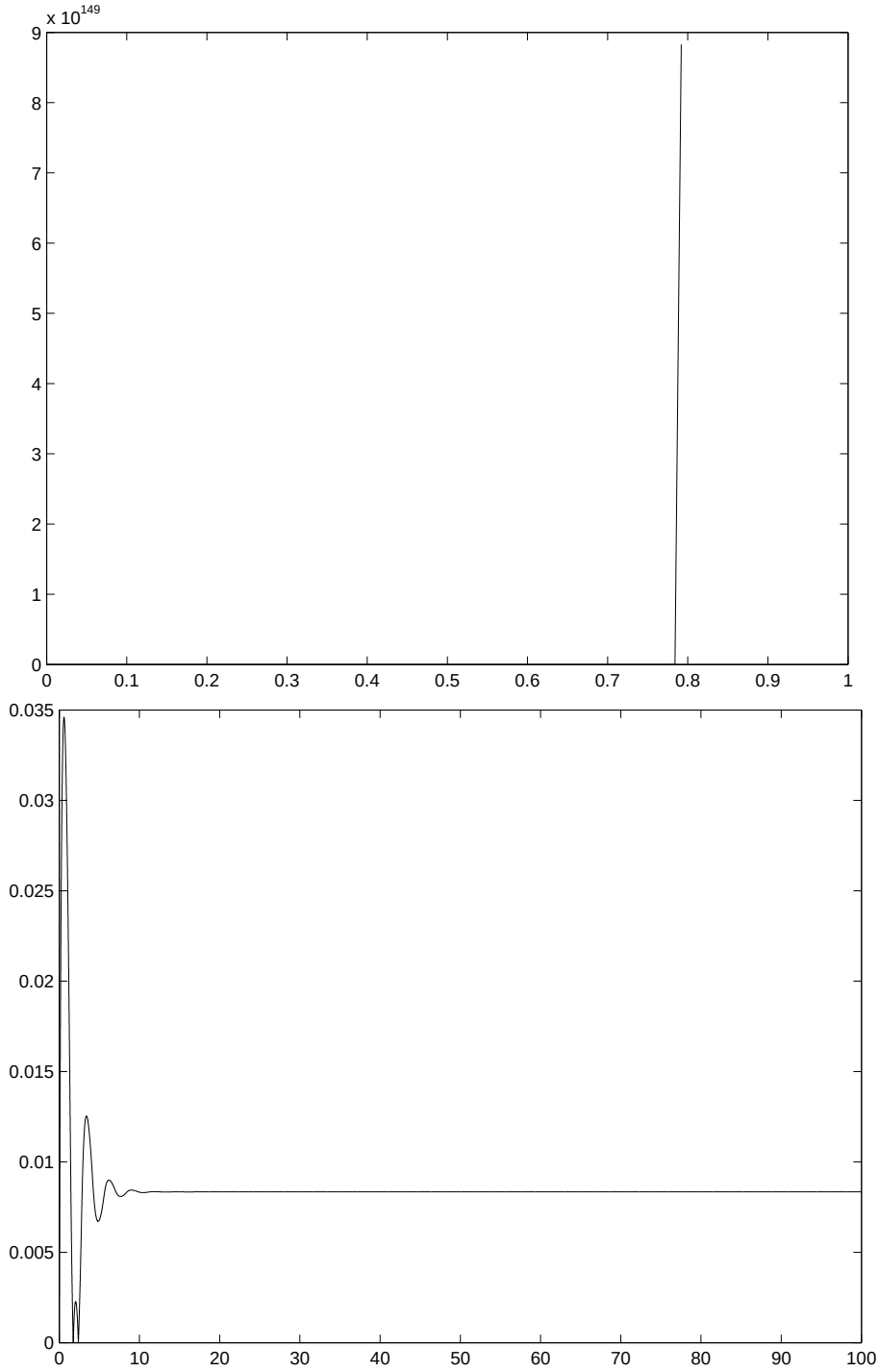

Figure 57: Crank-Nicholson method for Two-Variable Nonlinear Equation (a) $\alpha =$ .8; (b) $\alpha = .1$.

7 Future Research

Through the course of this thesis, we were able to investigate six different numerical methods. We narrowed the list to three that worked well for the nonlinear system. Of course, there is still much more to do. For the nonlinear system, we saw that the solutions were all stable up to 100 crossing times for a Courant factor of .1. However, due to the restraints of time, we were not able to look at longer crossing times. It would be interesting to see how long these different methods run before blowing up. Also, many of the methods were unstable for a Courant factor of .8. A next step would be to see at what Courant factor the solutions go from unstable to stable, up to a given number of crossing times. Perhaps the easiest way to find stability is through the von Neumann stability analysis we saw in Chapter 3. Unfortunately, the stability analysis would be much more complex as it involves more than one variable for the nonlinear system and is therefore beyond the scope of this research. We can also look at perturbation analyses for the various methods to determine the stability of a solution when a small perturbation is added.

Future research could also lead us in the direction of other initial and boundary conditions. We looked at an exponentially growing solution which goes to infinity rather quickly. It would be interesting to find one that doesn't. In [HKN], the authors considered a spatially constant solution, but this is almost too trivial to consider. As far as boundary conditions, we only considered periodic, outgoing, and Dirichlet throughout this paper and only Dirichlet boundary conditions in the

nonlinear system. There are many other types of boundary conditions, ranging in complexity, that could change the results of our numerical methods.

Moving away from this particular nonlinear solution, it is also important to model equations in higher dimension. We've only dealt with the one-dimensional wave equation and one of its nonlinear counterparts.

8 Appendix A: MATLAB Code for Advection Equation

```
%* Select numerical parameters (time step, grid spacing, etc.).
method = menu('Choose a numerical method:', 'FTCS','Lax', ...
    'Lax-Wendroff','Upwind','Leapfrog','Crank-Nicholson');
N=50; %Number of grid points
L = 1.; %Length of grid
h = L/N; %Grid spacing
c = 1; %Wave speed
alpha = input('Enter Courant factor: '); tau = h*alpha/c;
nStep=(input('Enter Number of Crossing Times: '))*L/(c*tau);
coeff = -c*tau/(2.*h); %Coeff used by all schemes
coefflw = 2*coeff^2; %Coeff used by L-W

%* Set initial and boundary conditions.
sigma = 0.1; % Width of the Gaussian pulse
k_wave = pi/sigma; % Wave number of the cosine
x = ((1:N)-1/2)*h -L/2; % Coordinates of grid points
%Initial condition is a Gaussian-cosine pulse
a = cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));
aold=cos(k_wave*(x+c*tau)) .* exp(-(x+c*tau).^2/(2*sigma^2));
%Use periodic boundary conditions
ip(1:(N-1)) = 2:N; ip(N)=1; % ip = i+1 with periodic B.C.
im(2:N) = 1:(N-1); im(1)=N; %im = i-1 with periodic B.C.
io(1:N) = 1:N;

%* Initialize plotting variables.
iplot = 1; %Plot counter
aplot(:,1) = a(:); % Record the inital state
tplot(1) = 0; % Record the initial time (t=0)
nplots = 50; %Desired number of plots
plotStep = nStep/nplots; %Number of steps between plots

%* Loop over desired number of steps.
for iStep=1:nStep %%MAIN LOOP%%
    %* Compute new values of wave amplitude using FTCS, Lax, or ...
        Lax-Wendroff method.
    if( method ==1 )  %%%FTCS method%%%
        a(io) = a(io) + coeff*(a(ip)-a(im));
    elseif( method == 2 )   %%%Lax method%%%
```

```
        a(io) = .5*(a(ip)+a(im)) + coeff*(a(ip)-a(im));
    elseif( method == 3 )   %%%Lax-Wendroff method)
        a(io) = a(io) + coeff*(a(ip)-a(im)) + ...
            coefflw*(a(ip)+a(im)-2*a(io));
    elseif(method == 4)  %%%Upwind method%%%
        a(io) = a(io) + 2*coeff*(a(io)-a(im));
    elseif(method == 5)   %%%Leapfrog method%%%
        anew(io) = aold(io) + 2*coeff*(a(ip)-a(im));
        aold = a;
        a = anew;
    else  %%%Crank-Nicholson method%%%
        anew(io) = a(io) + coeff*(a(ip)-a(im));
            %FTCS Method for guess
        for j=1:10
            anew(io) = a(io) + .5*coeff*(anew(ip) + a(ip) - ...
                anew(im) - a(im));
        end
        a=anew;
    end

    %*Periodically record a(t) for plotting.
    if( rem(iStep,plotStep) < 1) %Every plot_iter steps record
        iplot = iplot + 1;
        aplot(:,iplot) = a(:); % Record a(i) for plotting
        tplot(iplot) = tau*iStep;
        %fprintf('%g out of %g steps completed\n', iStep, nStep);
    end

    %* Compute L2 Norm (At Interior Points)
    atrue = cos(k_wave*x(2:N-1)) .* exp(-x(2:N-1).^2/(2*sigma^2));
    adiff = a(2:N-1) - atrue;
    aerr(iStep) = sqrt(sum(adiff(:).^2))/(N-2);
    timerr(iStep) = tau*iStep;

end

%*Plot the inital and final states.
figure(1); clf; % Clear figure 1 window and bring forward
plot(x,aplot(:,1),'-',x,a,':'); legend('Initial','Final');
xlabel('x'); ylabel('a(x,t)');

%* Plot the wave amplitude versus position and time
figure(2); clf; % Clear figure 2 window and bring forward
```

```
mesh(x,tplot,uplot);
xlabel('Position');
ylabel ('Time');
%zlabel('Amplitude');
view([20 60]); %Better view from this angle

%* Log Error vs Log time, used for FTCS
figure(3); clf; % Clear figure 3 window and bring forward
plot(log(timerr),log(aerr)); xlabel('log(t)'); ylabel('log(a(x,t)');
```

9 Appendix B: MATLAB Code for Wave Equation

For the wave equation, each method has its own file. Unlike with the advection equation where only one line was required to change the method, the wave equation changes a number of things. In what follows, we first present the basic template for the wave equation code. In the template, there are two parts that differ with each code. Thus, after we present the basic template, we will present those two parts for each individual method.

9.1 The Template

```
  % clear all;

%* Select numerical parameters.
N=100;
L= 1.; %system size
h= L/N; %Grid spacing
c= 1; %Wave speed
BC= input('Choose Boundary Condition- 1 for Periodic, 2 for
    outgoing, 3 for Dirichlet:');
alpha= input('Enter Courant factor: '); tau= h*alpha/c;
nStep=(input('Enter Number of Crossing Times:'))*L/(c*tau);

%%INSERT INDIVIDUAL CODE #1 HERE

%* Initialize plotting variables.
iplot = 1; %Plot counter
uplot(1,1:N) = u; % Record the inital state
tplot(1) = 0; % Record the initial time (t=0)
nplots = 50; %Desired number of plots
plotStep = nStep/nplots; %Number of steps between plots
tp = 0;

%* Loop over desired number of steps.
for iStep=1:nStep
```

```
    tp = tp+tau;
    t=iStep*tau;

%% INSERT INDIVIDAL CODE #2 HERE

    %*Periodically record a(t) for plotting.
    if( rem(iStep,plotStep) < 1) %Every plot_iter stpes record
        iplot = iplot + 1;
        uplot(iplot,1:N) = u; % Record a(i) for plotting
        tplot(iplot) = tau*iStep;
    end

    %* Compute L2 Norm (At Interior Points)
    if BC == 1
        for i=1:length(tp)
            if tp >= L/(2*c)
                tp = tp-(L/c);
            end
        end
        utrue = cos(k_wave*(x-c*tp)) .*exp(-(x-c*tp).^2/(2*sigma^2));
    elseif ( (BC == 2) | (BC == 3) )
         utrue = cos(k_wave*(x-c*t)) .*exp(-(x-c*t).^2/(2*sigma^2));
    end
    udiff = u - utrue;
    uerr(iStep) = sqrt(sum(udiff(:).^2))/N;
    timerr(iStep) = t;
end

%*Plot the inital and final states.
figure(1); clf; % Clear figure 1 window and bring forward
%plot(x,uplot(:,1),'-',x,u,'--'); legend('Initial','Final');
plot(x,u,'--'); legend('Final'); %see just final wave
xlabel('x'); ylabel('u(x,t)');
pause(1); %Pause 1 second between plots

%* Plot the wave amplitude versus position and time
figure(2); clf; % Clear figure 2 window and bring forward
mesh(x,tplot,uplot);
xlabel('Position');
ylabel ('Time');
zlabel('Amplitude');
view([20 60]); %Better view from this angle
```

```
%Error vs time for u
figure; plot(timerr,uerr);
```

9.2 Three-Variable System

9.2.1 Lax Method

Individual Code 1

```
%Initial condition is a Gaussian-cosine pulse
sigma = 0.1; % Width of the Gaussian pulse
k_wave = pi/sigma; % Wave number of the cosine
x = ((1:N)-1/2)*h -L/2; % Coordinates of grid points
u = cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));
w=(-k_wave)*sin(k_wave*x) .* exp(-x.^2/(2*sigma^2)) + ...
    (-x/sigma^2) .* cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));
v = -c*w;  %Right-moving Wave

%%Boundary Conditions
if BC == 1  %Periodic Boundary Conditions
    ip(1:(N-1)) = 2:N; ip(N)=1; % ip = i+1 with periodic B.C.
    im(2:N) = 1:(N-1); im(1)=N; %im = i-1 with periodic B.C.
    io(1:N) = 1:N;
elseif ( (BC == 2) | (BC == 3) ) %outgoing and Dirichlet B.C.
    ip = 3:N;
    im = 1:(N-2);
    io = 2:(N-1);
end
```

Individual Code 2

```
 unew(io) = .5*(u(ip) + u(im)) + (tau/2)*(v(ip)+v(im));
 vnew(io) = .5*(v(ip) + v(im)) + ((tau*c^2)/(2*h))*(w(ip) - w(im));
 wnew(io) = .5*(w(ip) + w(im)) + (tau/(2*h))*(v(ip) - v(im));

 %Boundary Conditions
 if BC == 2 %Outgoing Boundary Conditions
     unew(1) = u(2) + ((c*tau - h)/(c*tau + h))*(unew(2) - u(1));
     unew(N) = u(N-1) + ((c*tau - h)/(c*tau + h))*(unew(N-1) - u(N));
     vnew(1) = v(2) + ((c*tau - h)/(c*tau + h))*(vnew(2) - v(1));
     vnew(N) = v(N-1) + ((c*tau - h)/(c*tau + h))*(vnew(N-1) - v(N));
     wnew(1) = w(2) + ((c*tau - h)/(c*tau + h))*(wnew(2) - w(1));
     wnew(N) = w(N-1) + ((c*tau - h)/(c*tau + h))*(wnew(N-1) - w(N));
 elseif BC == 3 %Dirichlet Boundary Conditions
```

```
    unew(1)=cos(k_wave*(x(1)-c*t)).*exp(-(x(1)-c*t).^2/(2*sigma^2));
    unew(N)=cos(k_wave*(x(N)-c*t)).*exp(-(x(N)-c*t).^2/(2*sigma^2));
    wnew(1) = (-k_wave)*sin(k_wave*(x(1)-c*t)).* ...
           exp(-(x(1)-c*t).^2/(2*sigma^2)) + ...
           (-(x(1)-c*t)/sigma^2).*cos(k_wave*(x(1)-c*t)).* ...
           exp(-(x(1)-c*t).^2/(2*sigma^2));
    wnew(N) = (-k_wave)*sin(k_wave*(x(N)-c*t)).* ...
           exp(-(x(N)-c*t).^2/(2*sigma^2)) + ...
           (-(x(N)-c*t)/sigma^2).*cos(k_wave*(x(N)-c*t)).* ...
           exp(-(x(N)-c*t).^2/(2*sigma^2));
    vnew(1) = -c*wnew(1);
    vnew(N) = -c*wnew(N);
 end

 u = unew;
 v = vnew;
 w = wnew;
```

9.2.2 Lax-Wendroff Method

Individual Code 1

```
%Initial condition is a Gaussian-cosine pulse
sigma = 0.1; % Width of the Gaussian pulse
k_wave = pi/sigma; % Wave number of the cosine
x = ((1:N)-1/2)*h -L/2; % Coordinates of grid points
u = cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));
w=(-k_wave)*sin(k_wave*x) .* exp(-x.^2/(2*sigma^2)) + ...
    (-x/sigma^2) .* cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));
v = -c*w;

%%Boundary Conditions
if BC == 1  %Periodic Boundary Conditions
    ip(1:(N-1)) = 2:N; ip(N)=1; % ip = i+1 with periodic B.C.
    im(2:N) = 1:(N-1); im(1)=N; %im = i-1 with periodic B.C.
    io(1:N) = 1:N;
elseif ( (BC == 2) | (BC == 3))  %outgoing and Dirichlet B.C.
    ip = 3:N;
    im = 1:(N-2);
    io = 2:(N-1);
end
```

Individual Code 2

```
unew(io) = u(io) + tau*v(io) + ((c^2)*(tau^2)/(4*h))*(w(ip)-w(im));
vnew(io) = v(io) + ((tau*c^2)/(2*h))*(w(ip) - w(im)) + ...
   ((tau^2)*(c^2)/(2*(h^2)))*(v(ip)-2*v(io)+v(im));
wnew(io) = w(io) + (tau/(2*h))*(v(ip) - v(im)) + ...
   ((tau^2)*(c^2)/(2*(h^2)))*(w(ip)-2*w(io)+w(im));

%Boundary Conditions
if BC == 2 %Outgoing Boundary Conditions
    unew(1) = u(2) + ((c*tau - h)/(c*tau + h))*(unew(2) - u(1));
    unew(N) = u(N-1) + ((c*tau - h)/(c*tau + h))*(unew(N-1) - u(N));
    vnew(1) = v(2) + ((c*tau - h)/(c*tau + h))*(vnew(2) - v(1));
    vnew(N) = v(N-1) + ((c*tau - h)/(c*tau + h))*(vnew(N-1) - v(N));
    wnew(1) = w(2) + ((c*tau - h)/(c*tau + h))*(wnew(2) - w(1));
    wnew(N) = w(N-1) + ((c*tau - h)/(c*tau + h))*(wnew(N-1) - w(N));
elseif BC == 3 %Dirichlet Boundary Conditions
    unew(1)=cos(k_wave*(x(1)-c*t)).*exp(-(x(1)-c*t).^2/(2*sigma^2));
    unew(N)=cos(k_wave*(x(N)-c*t)).*exp(-(x(N)-c*t).^2/(2*sigma^2));
    wnew(1) = (-k_wave)*sin(k_wave*(x(1)-c*t)).* ...
           exp(-(x(1)-c*t).^2/(2*sigma^2)) + ...
           (-(x(1)-c*t)/sigma^2).*cos(k_wave*(x(1)-c*t)).* ...
           exp(-(x(1)-c*t).^2/(2*sigma^2));
    wnew(N) = (-k_wave)*sin(k_wave*(x(N)-c*t)).* ...
           exp(-(x(N)-c*t).^2/(2*sigma^2)) + ...
           (-(x(N)-c*t)/sigma^2).*cos(k_wave*(x(N)-c*t)).* ...
           exp(-(x(N)-c*t).^2/(2*sigma^2));
    vnew(1) = -c*wnew(1);
    vnew(N) = -c*wnew(N);
end

u = unew;
v = vnew;
w = wnew;
```

9.2.3 Leapfrog Method

Individual Code 1

```
%Initial condition is a Gaussian-cosine pulse
sigma = 0.1; % Width of the Gaussian pulse
k_wave = pi/sigma; % Wave number of the cosine
x = ((1:N)-1/2)*h -L/2; % Coordinates of grid points
xh = (x(2:end)+x(1:end-1))/2;% Half-step x values
```

```
u = cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));
w=(-k_wave)*sin(k_wave*xh) .* exp(-xh.^2/(2*sigma^2)) + ...
    (-xh/sigma^2) .* cos(k_wave*xh) .* exp(-xh.^2/(2*sigma^2));
v=-c*((-k_wave)*sin(k_wave*(x-(c*tau/2))).* ...
    exp(-(x-(c*tau/2)).^2/(2*sigma^2))+
    (-(x -(c*tau/2))/sigma^2).*cos(k_wave*(x -(c*tau/2))).* ...
    exp(-(x -(c*tau/2)).^2/(2*sigma^2)));

%%Boundary Conditions
if BC == 1  %Periodic Boundary Conditions
    io = 1:N;
    ipw(1:N-1) = 1:(N-1); ipw(N) = 1;
    imw(1) = N-1; imw(2:N) = 1:N-1;
    ipv = 2:N;
    imv = 1:N-1;
elseif ( (BC == 2) | (BC == 3))  %outgoing and Dirichlet B.C.
    io = 2:(N-1);
    ipw = 2:(N-1);
    imw = 1:(N-2);
    ipv = 2:N;
    imv = 1:(N-1);
end
```

Individual Code 2

```
 unew     = u     + tau*v;
 wnew     = w     + (tau/h)*(v(ipv) - v(imv));
 vnew(io) = v(io) + ((tau*c^2)/h)*(wnew(ipw) - wnew(imw));

 %Boundary Conditions
 if BC == 2 %Outgoing Boundary Conditions
     vnew(1) = v(2) + ((c*tau - h)/(c*tau + h))*(vnew(2) - v(1));
     vnew(N) = v(N-1) + ((c*tau - h)/(c*tau + h))*(vnew(N-1) - v(N));
 elseif BC == 3 %Dirichlet Boundary Conditions
     vnew(1) = -c*((-k_wave)*sin(k_wave*(x(1)-c*t)).* ...
            exp(-(x(1)-c*t).^2/(2*sigma^2)) + ...
            (-(x(1)-c*t)/sigma^2).*cos(k_wave*(x(1)-c*t)).* ...
            exp(-(x(1)-c*t).^2/(2*sigma^2)));
     vnew(N) = -c*((-k_wave)*sin(k_wave*(x(N)-c*t)).* ...
            exp(-(x(N)-c*t).^2/(2*sigma^2)) + ...
            (-(x(N)-c*t)/sigma^2).*cos(k_wave*(x(N)-c*t)).* ...
            exp(-(x(N)-c*t).^2/(2*sigma^2)));
 end
```

```
u = unew;
v = vnew;
w = wnew;
```

9.2.4 Crank-Nicholson Method

Individual Code 1

```
%Initial condition is a Gaussian-cosine pulse
sigma = 0.1; % Width of the Gaussian pulse
k_wave = pi/sigma; % Wave number of the cosine
x = ((1:N)-1/2)*h -L/2; % Coordinates of grid points
u = cos(k_wave*x) .* exp(-x.^2/(2*sigma^2)); uold =
cos(k_wave*(x+c*tau)) .* exp(-(x+c*tau).^2/(2*sigma^2)); w =
(-k_wave) * sin(k_wave*x) .* exp(-x.^2/(2*sigma^2)) + ...
    (-x/sigma^2) .* cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));
wold = w; v = -c*w; vold = v;

%%Boundary Conditions
if BC == 1  %Periodic Boundary Conditions
    ip(1:(N-1)) = 2:N; ip(N)=1; % ip = i+1 with periodic B.C.
    im(2:N) = 1:(N-1); im(1)=N; %im = i-1 with periodic B.C.
    io(1:N) = 1:N;
elseif ( (BC == 2) | (BC == 3))  %outgoing and Dirichlet B.C.
    ip = 3:N;
    im = 1:(N-2);
    io = 2:(N-1);
end
```

Individual Code 2

```
unew(io) = u(io) + (tau/2)*(v(ip)+v(im));
vnew(io) = v(io) + ((tau*c^2)/(2*h))*(w(ip) - w(im));
wnew(io) = w(io) + (tau/(2*h))*(v(ip) - v(im));

%Boundary Conditions
if BC == 2 %Outgoing Boundary Conditions
    unew(1) = u(2) + ((c*tau - h)/(c*tau + h))*(unew(2) - u(1));
    unew(N) = u(N-1) + ((c*tau - h)/(c*tau + h))*(unew(N-1) - u(N));
    vnew(1) = v(2) + ((c*tau - h)/(c*tau + h))*(vnew(2) - v(1));
    vnew(N) = v(N-1) + ((c*tau - h)/(c*tau + h))*(vnew(N-1) - v(N));
    wnew(1) = w(2) + ((c*tau - h)/(c*tau + h))*(wnew(2) - w(1));
    wnew(N) = w(N-1) + ((c*tau - h)/(c*tau + h))*(wnew(N-1) - w(N));
```

```
elseif BC == 3 %Dirichlet Boundary Conditions
    unew(1)=cos(k_wave*(x(1)-c*t)).*exp(-(x(1)-c*t).^2/(2*sigma^2));
    unew(N)=cos(k_wave*(x(N)-c*t)).*exp(-(x(N)-c*t).^2/(2*sigma^2));
    wnew(1) = (-k_wave)*sin(k_wave*(x(1)-c*t)).* ...
           exp(-(x(1)-c*t).^2/(2*sigma^2)) + ...
           (-(x(1)-c*t)/sigma^2).*cos(k_wave*(x(1)-c*t)).* ...
           exp(-(x(1)-c*t).^2/(2*sigma^2));
    wnew(N) = (-k_wave)*sin(k_wave*(x(N)-c*t)).* ...
           exp(-(x(N)-c*t).^2/(2*sigma^2)) + ...
           (-(x(N)-c*t)/sigma^2).*cos(k_wave*(x(N)-c*t)).* ...
           exp(-(x(N)-c*t).^2/(2*sigma^2));
    vnew(1) = -c*wnew(1);
    vnew(N) = -c*wnew(N);
end

for j=1:2
unew(io)= u(io) + (tau/2)*(vnew(io) + v(io));
vnew(io)= v(io) + ((tau*c^2)/(4*h))*(wnew(ip)+w(ip)-wnew(im)-w(im));
wnew(io)= w(io) + (tau/(4*h))*(vnew(ip) + v(ip) - vnew(im) - v(im));
end

u = unew;
v = vnew;
w = wnew;
```

9.3 Two-Variable System

9.3.1 Lax Method

Individual Code 1

```
%Initial condition is a Gaussian-cosine pulse
sigma = 0.1; % Width of the Gaussian pulse
k_wave = pi/sigma; % Wave number of the cosine
x = ((1:N)-1/2)*h -L/2; % Coordinates of grid points
u  = cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));

% Initial condition: v = -u for right-moving wave
%                    v = +u for left-moving wave
%                    v =  0 for split wave
v = -cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));

%%Boundary Conditions
if BC == 1  %Periodic Boundary Conditions
```

```
    ip(1:(N-1)) = 2:N; ip(N)=1; % ip = i+1 with periodic B.C.
    im(2:N) = 1:(N-1); im(1)=N; %im = i-1 with periodic B.C.
    io(1:N) = 1:N;
elseif ( (BC == 2) | (BC == 3) )  %outgoing and Dirichlet B.C.
    ip = 3:N;
    im = 1:(N-2);
    io = 2:(N-1);
end
```

Individual Code 2

```
unew(io) = .5*(u(ip) + u(im)) + ((tau*c)/(2*h))*(v(ip) - v(im));
vnew(io) = .5*(v(ip) + v(im)) + ((tau*c)/(2*h))*(u(ip) - u(im));

%Boundary Conditions
if BC == 2 %Outgoing Boundary Conditions
    unew(1) = u(2) + ((c*tau - h)/(c*tau + h))*(unew(2) - u(1));
    unew(N) = u(N-1) + ((c*tau - h)/(c*tau + h))*(unew(N-1) - u(N));
    vnew(1) = v(2) + ((c*tau - h)/(c*tau + h))*(vnew(2) - v(1));
    vnew(N) = v(N-1) + ((c*tau - h)/(c*tau + h))*(vnew(N-1) - v(N));
elseif BC == 3 %Dirichlet Boundary Conditions
    unew(1)=cos(k_wave*(x(1)-c*t)).*exp(-(x(1)-c*t).^2/(2*sigma^2));
    unew(N)=cos(k_wave*(x(N)-c*t)).*exp(-(x(N)-c*t).^2/(2*sigma^2));
    vnew(1) = -c*((-k_wave)*sin(k_wave*(x(1)-c*t)).* ...
           exp(-(x(1)-c*t).^2/(2*sigma^2)) + ...
           (-(x(1)-c*t)/sigma^2).*cos(k_wave*(x(1)-c*t)).* ...
           exp(-(x(1)-c*t).^2/(2*sigma^2)));
    vnew(N) = -c*((-k_wave)*sin(k_wave*(x(N)-c*t)).* ...
           exp(-(x(N)-c*t).^2/(2*sigma^2)) + ...
           (-(x(N)-c*t)/sigma^2).*cos(k_wave*(x(N)-c*t)).* ...
           exp(-(x(N)-c*t).^2/(2*sigma^2)));
end

u = unew;
v = vnew;
```

9.3.2 Lax-Wendroff Method

Individual Code 1

```
%Initial condition is a Gaussian-cosine pulse
sigma = 0.1; % Width of the Gaussian pulse
k_wave = pi/sigma; % Wave number of the cosine
x = ((1:N)-1/2)*h -L/2; % Coordinates of grid points
```

```
u  = cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));

% Initial condition: v = -u for right-moving wave
%                    v = +u for left-moving wave
%                    v =  0 for split wave
v = -cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));

%%Boundary Conditions
if BC == 1  %Periodic Boundary Conditions
    ip(1:(N-1)) = 2:N; ip(N)=1; % ip = i+1 with periodic B.C.
    im(2:N) = 1:(N-1); im(1)=N; %im = i-1 with periodic B.C.
    io(1:N) = 1:N;
elseif ( (BC == 2) | (BC == 3))  %outgoing and Dirichlet B.C.
    ip = 3:N;
    im = 1:(N-2);
    io = 2:(N-1);
end
```

Individual Code 2

```
 unew(io) = u(io) + ((tau*c)/(2*h))*(v(ip) - v(im)) + ...
    ((c^2)*(tau^2)/(2*h^2))*(u(ip) - 2*u(io) + u(im));
 vnew(io) = v(io) + ((tau*c)/(2*h))*(u(ip) - u(im)) + ...
    ((c^2)*(tau^2)/(2*h^2))*(v(ip) - 2*v(io) + v(im));

 %Boundary Conditions
 if BC == 2 %Outgoing Boundary Conditions
     unew(1) = u(2) + ((c*tau - h)/(c*tau + h))*(unew(2) - u(1));
     unew(N) = u(N-1) + ((c*tau - h)/(c*tau + h))*(unew(N-1) - u(N));
     vnew(1) = v(2) + ((c*tau - h)/(c*tau + h))*(vnew(2) - v(1));
     vnew(N) = v(N-1) + ((c*tau - h)/(c*tau + h))*(vnew(N-1) - v(N));
 elseif BC == 3 %Dirichlet Boundary Conditions
     unew(1)=cos(k_wave*(x(1)-c*t)).*exp(-(x(1)-c*t).^2/(2*sigma^2));
     unew(N)=cos(k_wave*(x(N)-c*t)).*exp(-(x(N)-c*t).^2/(2*sigma^2));
     vnew(1) = -c*((-k_wave)*sin(k_wave*(x(1)-c*t)).* ...
            exp(-(x(1)-c*t).^2/(2*sigma^2)) + ...
            (-(x(1)-c*t)/sigma^2).*cos(k_wave*(x(1)-c*t)).* ...
            exp(-(x(1)-c*t).^2/(2*sigma^2)));
     vnew(N) = -c*((-k_wave)*sin(k_wave*(x(N)-c*t)).* ...
            exp(-(x(N)-c*t).^2/(2*sigma^2)) + ...
            (-(x(N)-c*t)/sigma^2).*cos(k_wave*(x(N)-c*t)).* ...
            exp(-(x(N)-c*t).^2/(2*sigma^2)));
 end
```

```
 u = unew;
 v = vnew;
```

9.3.3 Leapfrog Method

Individual Code 1

```
%Initial condition is a Gaussian-cosine pulse
sigma = 0.1; % Width of the Gaussian pulse
k_wave = pi/sigma; % Wave number of the cosine
x = ((1:N)-1/2)*h -L/2; % Coordinates of grid points
xh(1:N-1) = (x(2:end)+x(1:end-1))/2;% Half-step x values
u  = cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));

% Initial condition: v = -u for right-moving wave
%                    v = +u for left-moving wave
%                    v =  0 for split wave
v = -cos(k_wave*(xh -(c*tau/2))) .* exp(-(xh
-(c*tau/2)).^2/(2*sigma^2));

%%Boundary Conditions
if BC == 1  %Periodic Boundary Conditions
    ipu = 2:N;
    imu = 1:N-1;
    iou = 1:N;
    ipv(1:N-1) = 1:N-1; ipv(N) = 1;
    imv(1) = N-1; imv(2:N) = 1:N-1;
elseif ( (BC == 2) | (BC == 3)  %outgoing and Dirichlet B.C.
    ipu = 2:N;
    imu = 1:(N-1);
    iou = 2:(N-1);
    ipv = 2:(N-1);
    imv = 1:(N-2);
end
```

Individual Code 2

```
 unew(iou) = u(iou) + (c*tau/h)*(v(ipv) - v(imv));

 %Boundary Conditions
 if BC == 2 %Outgoing Boundary Conditions
     unew(1) = u(2) + ((c*tau - h)/(c*tau + h))*(unew(2) - u(1));
     unew(N) = u(N-1) + ((c*tau - h)/(c*tau + h))*(unew(N-1) - u(N));
```

```
 elseif BC == 3 %Dirichlet Boundary Conditions
     unew(1)=cos(k_wave*(x(1)-c*t)).*exp(-(x(1)-c*t).^2/(2*sigma^2));
     unew(N)=cos(k_wave*(x(N)-c*t)).*exp(-(x(N)-c*t).^2/(2*sigma^2));
 end

 u = unew;
 vnew     = v     + (c*tau/h)*(u(ipu)  - u(imu) );
 v = vnew;
```

9.3.4 Crank-Nicholson Method

Individual Code 1

```
%Initial condition is a Gaussian-cosine pulse
sigma = 0.1; % Width of the Gaussian pulse
k_wave = pi/sigma; % Wave number of the cosine
x = ((1:N)-1/2)*h -L/2; % Coordinates of grid points
u  = cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));

% Initial condition: v = -u for right-moving wave
%                    v = +u for left-moving wave
%                    v =  0 for split wave
v = -cos(k_wave*x) .* exp(-x.^2/(2*sigma^2));

%%Boundary Conditions
if BC == 1  %Periodic Boundary Conditions
    ip(1:(N-1)) = 2:N; ip(N)=1; % ip = i+1 with periodic B.C.
    im(2:N) = 1:(N-1); im(1)=N; %im = i-1 with periodic B.C.
    io(1:N) = 1:N;
elseif ( (BC == 2) | (BC == 3))  %outgoing and Dirichlet B.C.
    ip = 3:N;
    im = 1:(N-2);
    io = 2:(N-1);
end
```

Individual Code 2

```
 unew(io) = u(io) + ((tau*c)/(2*h))*(v(ip) - v(im));
 vnew(io) = v(io) + ((tau*c)/(2*h))*(u(ip) - u(im));

 %Boundary Conditions
 if BC == 2 %Outgoing Boundary Conditions
     unew(1) = u(2) + ((c*tau - h)/(c*tau + h))*(unew(2) - u(1));
     unew(N) = u(N-1) + ((c*tau - h)/(c*tau + h))*(unew(N-1) - u(N));
```

```
    vnew(1) = v(2) + ((c*tau - h)/(c*tau + h))*(vnew(2) - v(1));
    vnew(N) = v(N-1) + ((c*tau - h)/(c*tau + h))*(vnew(N-1) - v(N));
elseif BC == 3 %Dirichlet Boundary Conditions
    unew(1)=cos(k_wave*(x(1)-c*t)).*exp(-(x(1)-c*t).^2/(2*sigma^2));
    unew(N)=cos(k_wave*(x(N)-c*t)).*exp(-(x(N)-c*t).^2/(2*sigma^2));
    vnew(1) = -c*((-k_wave)*sin(k_wave*(x(1)-c*t)).* ...
           exp(-(x(1)-c*t).^2/(2*sigma^2)) + ...
           (-(x(1)-c*t)/sigma^2).*cos(k_wave*(x(1)-c*t)).* ...
           exp(-(x(1)-c*t).^2/(2*sigma^2)));
    vnew(N) = -c*((-k_wave)*sin(k_wave*(x(N)-c*t)).* ...
           exp(-(x(N)-c*t).^2/(2*sigma^2)) + ...
           (-(x(N)-c*t)/sigma^2).*cos(k_wave*(x(N)-c*t)).* ...
           exp(-(x(N)-c*t).^2/(2*sigma^2)));
end

for j=1:2
unew(io)= u(io) + ((tau*c)/(4*h))*(vnew(ip)+v(ip)-vnew(im)-v(im));
vnew(io)= v(io) + ((tau*c)/(4*h))*(unew(ip)+u(ip)-unew(im)-u(im));
end

u = unew;
v = vnew;
```

10 Appendix C: MATLAB Code for Nonlinear Wave Equation

We will again use the template method seen in Appendix B.

10.1 The Template

```
clear all; close all;

%* Select numerical parameters (time step, grid spacing, etc.).
N= 100;
L= 1.; %system size
h= L/N; %Grid spacing
c= 1; %Wave speed
alpha= input('Enter Courant Factor: ');
tau= h*alpha/c;
nStep=(input('Enter Number of Crossing Times: '))*L/(c*tau);

%* Set initial and boundary conditions.
sigma = 0.1; % Width of the Gaussian pulse
k_wave = pi/sigma; % Wave number of the cosine
x = ((1:N)-1/2)*h -L/2; % Coordinates of grid points

ip = 3:N; im = 1:(N-2); io = 2:(N-1);

INSERT INDIVIDUAL CODE #1

%* Initialize plotting variables.
iplot = 1; %Plot counter
gplot(:,1) = g(:); % Record the inital state
kplot(:,1) = k(:); % Record the inital state
tplot(1) = 0; % Record the initial time (t=0)
nplots = 50; %Desired number of plots
plotStep = nStep/nplots; %Number of steps between plots

%* Loop over desired number of steps.
for iStep=1:nStep %%MAIN LOOP%%
    t = tau*iStep;

INSERT INDIVIDUAL CODE #2
```

```
    %*Periodically record a(t) for plotting.
    if( rem(iStep,plotStep) < 1) %Every plot_iter stpes record
        iplot = iplot + 1;
        gplot(:,iplot) = g(:); % Record a(i) for plotting
        tplot(iplot) = tau*iStep;
    end

     %* Compute Relative Error in g
     gtrue = exp(x + t/sqrt(2));
     gerr(iStep) = max(abs((g-gtrue)/gtrue));
     timerr(iStep) = t;
 end

%Error vs time
figure; plot(timerr,gerr);
```

10.2 Three-Variable System

10.2.1 Lax-Wendroff Method

Individual Code 1

```
        g = exp(x).*ones(size(x));
        k = +g/sqrt(2);
        w = g;
        gnew = zeros(size(g));
        knew = zeros(size(k));
        wnew = zeros(size(w));
```

Individual Code 2

```
        wx = (w(ip) - w(im))/(2*h);
        wxx = (w(ip) - 2*w(io) + w(im))/h^2;
        kx = (k(ip) - k(im))/(2*h);
        kxx = (k(ip) - 2*k(io) + k(im))/h^2;
        gxx = wx;

        gt = k(io);
        gtt = gxx - k(io).^2 ./ g(io);

        kt = gtt;
        ktt = kxx - (2*k(io).*kt ./ g(io))  + k(io).^3 ./ g(io).^2;

        wt = kx;
```

```
wtt = wxx - (2*k(io).*kx ./g(io)) + (k(io)./g(io)).^2 .* w(io);

gnew(io) = g(io) + tau * gt + 0.5*tau*tau * gtt;
knew(io) = k(io) + tau * kt + 0.5*tau*tau * ktt;
wnew(io) = w(io) + tau * wt + 0.5*tau*tau * wtt;

%Boundary Conditions
    gnew(1) = exp(x(1) + t/sqrt(2));
    gnew(N) = exp(x(N) + t/sqrt(2));
    knew(1) = +gnew(1)/sqrt(2);
    knew(N) = +gnew(N)/sqrt(2);
    wnew(1) = gnew(1);
    wnew(N) = gnew(N);


k = knew;
g = gnew;
w = wnew;
```

10.2.2 Leapfrog Method

Individual Code 1

```
gold = exp(x);
kold = gold/sqrt(2);
wold = gold;
g = exp(x + tau/sqrt(2));
k = g/sqrt(2);
w = g;
gnew = zeros(size(g));
knew = zeros(size(k));
wnew = zeros(size(w));
```

Individual Code 2

```
wx = (w(ip)-w(im))/(2*h);
kx = (k(ip)-k(im))/(2*h);

gt = k(io);
kt = wx - k(io).^2 ./ g(io);
wt = kx;

gnew(io) = gold(io) + 2*tau * gt;
knew(io) = kold(io) + 2*tau * kt;
```

```
wnew(io) = wold(io) + 2*tau * wt;

%Boundary Conditions
    gnew(1) = exp(x(1) + t/sqrt(2));
    gnew(N) = exp(x(N) + t/sqrt(2));
    knew(1) = gnew(1)/sqrt(2);
    knew(N) = gnew(N)/sqrt(2);
    wnew(1) = gnew(1);
    wnew(N) = gnew(N);

kold = k; gold = g; wold = w;
k = knew; g = gnew; w = wnew;
```

10.2.3 Crank-Nicholson Method

Individual Code 1

```
g = exp(x).*ones(size(x));
k = +g/sqrt(2);
w = g;
gnew = zeros(size(g));
knew = zeros(size(k));
wnew = zeros(size(w));
```

Individual Code 2

```
wx = (w(ip) - w(im))/(2*h);
kx = (k(ip) - k(im))/(2*h);
gxx = wx;
gtt = gxx - k(io).^2 ./ g(io);

gt = k(io);
kt = gtt;
wt = kx;

%Initial FTCS Step
gnew(io) = g(io) + tau * gt;
knew(io) = k(io) + tau * kt;
wnew(io) = w(io) + tau * wt;

%Boundary Conditions
    gnew(1) = exp(x(1) + t/sqrt(2));
    gnew(N) = exp(x(N) + t/sqrt(2));
    knew(1) = +gnew(1)/sqrt(2);
```

```
    knew(N) = +gnew(N)/sqrt(2);
    wnew(1) = gnew(1);
    wnew(N) = gnew(N);


wxnew = (wnew(ip) - wnew(im))/(2*h);
kxnew = (knew(ip) - knew(im))/(2*h);
gxxnew = wxnew;
gttnew = gxxnew - knew(io).^2 ./ gnew(io);

gtnew = knew(io);
ktnew = gttnew;
wtnew = kxnew;

%2 Crank Nicholson Iterations
for j=1:2;
    gnew(io) = g(io) + (tau/2) * (gtnew + gt);
    knew(io) = k(io) + (tau/2) * (ktnew + kt);
    wnew(io) = w(io) + (tau/2) * (wtnew + wt);
end

k = knew;
g = gnew;
w = wnew;
```

10.3 Two-Variable System

10.3.1 Lax-Wendroff Method

Individual Code 1

```
g = exp(x).*ones(size(x));
k = +g/sqrt(2);
gnew = zeros(size(g));
knew = zeros(size(k));
```

Individual Code 2

```
gxx = (g(ip) + g(im) -2*g(io))/(h^2);
kxx = (k(ip) + k(im)- 2*k(io))/(h^2);

gt  = k(io);
kt  = gxx - (k(io).^2)./g(io);
```

```
gtt = kt;
ktt = kxx - (2*k(io).*kt./g(io)) + (k(io).^3)./(g(io).^2);

knew(io) = k(io) + tau*kt + ((tau^2)/2)*ktt;
gnew(io) = g(io) + tau*gt + ((tau^2)/2)*gtt;

%Boundary Conditions
    gnew(1) = exp(x(1) + t/sqrt(2));
    gnew(N) = exp(x(N) + t/sqrt(2));
    knew(1) = +gnew(1)/sqrt(2);
    knew(N) = +gnew(N)/sqrt(2);

k = knew;
g = gnew;
```

10.3.2 Leapfrog Method

Individual Code 1

```
gold = exp(x);
kold = gold/sqrt(2);
g = exp(x + tau/sqrt(2));
k = g/sqrt(2);
gnew = zeros(size(g));
knew = zeros(size(k));
```

Individual Code 2

```
gxx = (g(ip) + g(im) -2*g(io))/(h^2);
gtt = gxx - k(io).^2 ./ g(io);

gt = k(io);
kt = gtt;

gnew(io) = gold(io) + 2*tau * gt;
knew(io) = kold(io) + 2*tau * kt;

%Boundary Conditions
    gnew(1) = exp(x(1) + t/sqrt(2));
    gnew(N) = exp(x(N) + t/sqrt(2));
    knew(1) = gnew(1)/sqrt(2);
    knew(N) = gnew(N)/sqrt(2);

kold = k; gold = g;
```

```
k = knew; g = gnew;
```

10.3.3 Crank-Nicholson Method

Individual Code 1

```
g = exp(x).*ones(size(x));
k = +g/sqrt(2);
gnew = zeros(size(g));
knew = zeros(size(k));
```

Individual Code 2

```
gxx = (g(ip) + g(im) -2*g(io))/(h^2);
gtt = gxx - k(io).^2 ./ g(io);

gt = k(io);
kt = gtt;

%Initial Lax Step
gnew(io) = g(io) + tau * gt;
knew(io) = g(io) + tau * kt;

%Boundary Conditions
    gnew(1) = exp(x(1) + t/sqrt(2));
    gnew(N) = exp(x(N) + t/sqrt(2));
    knew(1) = gnew(1)/sqrt(2);
    knew(N) = gnew(N)/sqrt(2);

gxxnew = (gnew(ip) + gnew(im) -2*gnew(io))/(h^2);
gttnew = gxxnew - (knew(io).^2) ./ gnew(io);

gtnew = knew(io);
ktnew = gttnew;

%2 Crank Nicholson Iterations
for j=1:2;
    gnew(io) = g(io) + (tau/2) * (gtnew + gt);
    knew(io) = k(io) + (tau/2) * (ktnew + kt);
end

k = knew;
g = gnew;
```

References

[BW] Barry C. Barish and Rainer Weiss, "LIGO and the Detection of Gravitational Waves," *Physics Today*, Volume 52, No. 10, 1999.

[HKN] Jakob Hansen, Alexei Khokhlov, and Igor Novikov, "Properties of four numerical schemes applied to a scalar nonlinear scalar wave equation with a GR-Type nonlinearity,"

[KT] Kip S. Thorne, "Probing Black Holes and Relativisitic Stars with Gravitational Waves," arXiv: gr-qc/9706079, 1997.

[NM] John H. Mathews and Krtix D. Fink, *Numerical Methods Using MATLAB*, Prentice-Hall, Inc. , 1999.

[NR] *Numerical Recipes in Fortran 77: The Art of Scientific Computing*, Cambridge University Press, 1992.

[T] Saul A. Teukolsky, "Stability of the Iterated Crank-Nicholson Method in Numerical Relativity," *Physical Review D*, Volume 61, American Physical Society, 2000.

[W] Clifford M. Will, "Gravitational Radiation and the Validity of General Relativity," *Physics Today, Volume 52, No. 10, 1999.*